**SPRING INTERVIEW QUESTIONS BY ASHOK**

[**Spring Framework Tutorial - How to call Stored Procedures from Java using IN and OUT parameter example**](http://javarevisited.blogspot.in/2013/04/spring-framework-tutorial-call-stored-procedures-from-java.html)

Spring Framework provides excellent support to call stored procedures from Java application. In fact there are multiple ways to call stored procedure in Spring Framework, e.g. you can use one of the query() method from JdbcTemplate to callstored procedures, or you can extend [abstract class](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html)StoredProcedure to call stored procedures from Java. In this Java Spring tutorial, we will see second approach to call stored procedure. It's more [object oriented](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html), but same time requires more coding. StoredProcedure

class allows you to declare IN and OUT parameters and call stored procedure using its variousexecute() method, which has protected access and can only be called from sub class. I personally prefer to implementStoredProcedure class as [Inner class](http://javarevisited.blogspot.sg/2012/12/inner-class-and-nested-static-class-in-java-difference.html), if its tied up with one of [DAO Object](http://javarevisited.blogspot.com/2013/01/data-access-object-dao-design-pattern-java-tutorial-example.html), e.g. in this case it nicely fit inside EmployeeDAO. Then you can provide convenient method to wrap stored procedure calls. In order to demonstrate, how to call stored procedures from spring based application, we will first create a simple stored proc using MySQL database, as shown below.

MySQL Stored procedure

We will use following stored procedure for this example. This is created in MySQL database and accept an input parameter IN, which is employeeId and return name of employee using its output parameter called, name.

Spring Stored Procedure example and Configurations

Here is complete code example of how to call stored procedure from Spring framework. In this example, we have extended [abstract class](http://java67.blogspot.com/2013/02/can-abstract-class-have-constructor-in-java.html) StoredProcedure in our class called, EmployeeSP. This is declared as[nested class](http://java67.blogspot.com/2012/10/nested-class-java-static-vs-non-static-inner.html) inside EmployeeDAO because its only used by this class, if your stored procedure is used my multiple DAO classes, than you can also make it a top level class. If you look at [constructor](http://javarevisited.blogspot.com/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) of EmployeeSP, it calls super class constructor and passes datasource and name of database stored procedure. We have also declared two stored procedure parameters, one is IN parameter

id, and other is OUT parameter. Input to stored procedure is passed using IN parameter, and output from stored procedure is read using OUT parameter. Your stored procedure can have multiple IN and OUT parameter. StoredProcedure class also provide several execute() methods, which can be invoked to call stored procedure and get result. It return result as [Map](http://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html), where key is OUT parameter, and value is result of stored procedure. Here is the code for DAO class and stored procedure along with Spring Configuration file, since Spring framework is based on principle of

[dependency Injection and Inversion of control](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html), this file is required to create and manage object.

That's all on **How to call stored procedure from Java application using Spring Framework**. As discussed in 10 JDBC best practices for Java Programmer, JDBC API provides more straightforward approach using CallableStatement, but Spring'sStoredProcedure class is also easy to use. You can also explore calling stored procedure, directly using JdbcTemplate in Spring.

Question 2: Explain the Spring Bean-LifeCycle.

Ans: Spring framework is based on IOC so we call it as IOC container also So Spring beans reside inside the IOC container. Spring beans are nothing but Plain old java object (POJO).

Following steps explain their life cycle inside the container.

1. The container will look the bean definition inside configuration file (e.g. bean.xml).

2 using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.

3. If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean’s ID.  
4. If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.  
5. If there are any BeanPostProcessors associated with the bean, their post-ProcessBeforeInitialization() methods will be called before the properties for the Bean are set.

6. If an init() method is specified for the bean, it will be called.  
7. If the Bean class implements the DisposableBean interface, then the method destroy() will be called when the Application no longer needs the bean reference.

8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.

**Question 6: What is the difference between singleton and prototype bean**

Ans: This is another popular *spring interview questions* and an important concept to understand. Basically, a bean has scopes which define their existence on the application

Singleton: means single bean definition to a single object instance per Spring IOC container.  
**Prototype**: means a single bean definition to any number of object instances.

Whatever beans we defined in spring framework are singleton beans. There is an attribute in bean tag named ‘singleton’ if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default, it is set to true. So, all the beans in spring framework are by default singleton beans.

<bean id="createNewStock" class="springexample.stockMarket.CreateNewStockAccont" **singleton=”false”**>  
 <property name="newBid"/>   
 </bean>

**JDBC Database connection pool in Spring Framework – How to Setup Example**  
Setting up JDBC Database Connection Pool in Spring framework is easy for any Java application, just matter of changing few configuration in spring configuration file.If you are writing core java application and not running on any web or application server like Tomcat or Weblogic, Managing Database connection pool using **Apache Commons DBCP** and Commons Pool along-with Spring framework is nice choice but if you have luxury of having web server and managed J2EE Container, consider using **Connection pool managed by J2EE server**

those are better option in terms of maintenance, flexibility and also help to prevent [java.lang.OutofMemroyError:PermGen Space in tomcat](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html) by avoiding loading of JDBC driver in web-app class-loader, Also keeping JDBC connection pool information in Server makes it easy to change or include settings for JDBC over SSL. In this article we will see **how to setup Database connection pool in spring framework** using Apache commons DBCP and commons pool.jar

Spring Example JDBC Database Connection Pool

Spring framework provides convenient JdbcTemplate class for performing all Database related operation. if you are not using Hibernate than using Spring's JdbcTemplate is good option. JdbcTemplate requires a DataSource which is javax.sql.DataSource implementation and you can get this directly using [spring bean](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html) configuration or by using **JNDI** if you are using [J2EE web server or application server](http://javarevisited.blogspot.sg/2012/05/5-difference-between-application-server.html) for managing Connection Pool. See *How to setup JDBC connection Pool in tomcat and Spring* for JNDI based connection pooling for more details. In order to setup

Data source you will require following configuration in your applicationContext.xml (spring configuration) file:

//Datasource connection settings in Spring  
<bean id="springDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close" >  
 <property name="url" value="jdbc:oracle:thin:@localhost:1521:SPRING\_TEST" />  
 <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />  
 <property name="username" value="root" />  
 <property name="password" value="root" />  
 <property name="removeAbandoned" value="true"/>  
 <property name="initialSize" value="20" />  
 <property name=

"maxActive" value="30" />  
</bean>  
  
//Dao class configuration in spring  
 <bean id="EmployeeDatabaseBean" class="com.test.EmployeeDAOImpl">  
 <property name="dataSource" ref="springDataSource"/>  
 </bean>

Below configuration of DBCP connection pool will create 20 database connection as initialSize is 20 and goes up to 30 Database connection if required as maxActive is 30. you can customize your database connection pool by using different properties provided by Apache DBCP library. Above example is creating connection pool with Oracle 11g database and we are using oracle.jdbc.driver.OracleDriver comes along with ojdbc6.jar or ojdbc6\_g.jar , to learn more about [how to connect Oracle database from Java program](http://javarevisited.blogspot.sg/2012/04/java-program-to-connect-oracle-database.html)

H[ow do you ensure that a thread executes after another?](https://stackoverflow.com/questions/31191268/how-do-you-ensure-that-a-thread-executes-after-another)

Your proposed code won't work. t3 could finish before t2 even starts. To guarantee they run in sequence, you need to to this:

final Thread t3 = new Thread(new T3()); // assume T3 is a Runnable

final Thread t2 = new Thread(new T2());

final Thread t1 = new Thread(new T1());

t1.start();

t1.join();

t2.start();

t2.join();

t3.start();

But then using multiple threads is pointless and wasteful. You would achieve the same effect with a single thread:

new T1().run();

new T2().run();

new T3().run();