## he answer and explanation to finding the nth highest salary in SQL

|  |
| --- |
|  |

Here we will present one possible answer to finding the nth highest salary first, and the explanation of that answer after since it’s actually easier to understand that way. Note that the first answer we present is actually not optimal from a performance standpoint since it uses a subquery, but we think that it will be interesting for you to learn about because you might just learn something new about SQL. If you want to see the more optimal solutions first, you can skip down to the sections that says “Find the nth highest salary without a subquery” instead.

The SQL below will give you the correct answer – but you will have to plug in an actual value for N of course. This SQL to find the Nth highest salary should work in SQL Server, MySQL, DB2, Oracle, Teradata, and almost any other RDBMS:

SELECT \* /\*This is the outer query part \*/

FROM Employee Emp1

WHERE (N-1) = ( /\* Subquery starts here \*/

SELECT COUNT(DISTINCT(Emp2.Salary))

FROM Employee Emp2

WHERE Emp2.Salary > Emp1.Salary)

## How does the query above work?

The query above can be quite confusing if you have not seen anything like it before – pay special attention to the fact that “Emp1″ appears in both the subquery (also known as an inner query) and the “outer” query. The outer query is just the part of the query that is not the subquery/inner query – both parts of the query are clearly labeled in the comments.

## The subquery is a correlated subquery

The subquery in the SQL above is actually a specific type of subquery known as a***correlated*** subquery. The reason it is called a correlated subquery is because the the subquery uses a value from the outer query in it’s WHERE clause. In this case that value is the Emp1 table alias as we pointed out earlier. A normal subquery can be run***independently*** of the outer query, but a *correlated* subquery can **NOT** be run independently of the outer query. If you want to read more about the differences between correlated and uncorrelated subqueries you can go here: [Correlated vs Uncorrelated Subqueries](http://www.programmerinterview.com/index.php/database-sql/correlated-vs-uncorrelated-subquery/).

 ====================================

**2. How Generics works in Java ? What is type erasure ?**

This is one of better interview question in Generics. Generics is implemented using **Type erasure**, compiler erases all type related information during compile time and no type related information is available during runtime. for example List<String> is represented by only [List](http://www.blogger.com/goog_1304192070)at runtime. This was done to ensure binary compatibility with the libraries which were developed prior to Java 5. you don't have access to Type argument at runtime and **Generic type** is translated to**Raw type** by compiler during [runtime](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html). you can get lot of follow up question based on this Generic interview question based upon your response e.g. *Why Generics is implemented using Type erasure* or presenting some invalid generic code which results in compiler error. read my post [How Generics works in Java](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html) for more details

**3. What is Bounded and Unbounded wildcards in Generics ?**

This is another very [popular Java interview questions](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) on Generics. Bounded Wildcards are those which impose bound on Type. there are two kinds of Bounded wildcards <? extends T> which impose an upper bound by ensuring that type must be sub class of T and <? super T> where its imposing lower bound by ensuring Type must be super class of T. This Generic Type must be instantiated with Type within bound otherwise it will result in compilation error. On the other hand <?> represent and unbounded type because <?> can be replace with any Type. See more on my post [differences between Bounded and Unbounded wildcards in Generics](http://javarevisited.blogspot.com/2012/04/what-is-bounded-and-unbounded-wildcards.html).

Merge: suppose we create a session and load an object. Now object in session cache. If we close the session at this point and we edit state of object and tried to save using update() it will throw exception. To make object persistent we need to open another session. Now we load same object again in current session. So if we want to update present object with previous object changes we have to use merge() method. Merge method will merge changes of both states of object and will save in database.   
  
Update: If we are dealing with any object in same session we should use update() or saveOrUpdate() method.

Merge: merge is like combining records from more than one table(while retreving records from tables based on some conditions)   
  
  
Update: Update is like edit . use to change the value of record..   
  
  
Update():- if you are sure that the session does not contains an already persistent instance with the same identifier,then use update to save the data in hibernate   
  
Merge():-if you want to save your modificatiions at any time with out knowing abot the state of an session, then use merge() in hibernate.

//Sample struts-config.xml  
<?xml version="1.0" encoding="ISO-8859-1" ?>   
<!DOCTYPE struts-config PUBLIC   
"-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"   
"<http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd>">   
<struts-config> ***Form bean Definitions***  
<form-beans>   
<form-bean name="CustomerForm"   
type="mybank.example.CustomerForm"/>   
<form-bean name="LogonForm"   
type="mybank.example.LogonForm"/>   
</form-beans> ***Global Forward Definitions***  
<global-forwards>   
<forward name="logon" path="/logon.jsp"/>   
<forward name="logoff" path="/logoff.do"/>   
</global-forwards> ***Action Mappings***  
<action-mappings>   
<action path="/submitDetailForm"   
type="mybank.example.CustomerAction"   
name="CustomerForm"   
scope="request"   
validate="true"   
input="/CustomerDetailForm.jsp">   
<forward name="success"   
path="/ThankYou.jsp"   
redirect=”true” />   
<forward name="failure"   
path="/Failure.jsp" />   
</action>   
<action path=”/logoff” parameter=”/logoff.jsp”   
type=”org.apache.struts.action.ForwardAction” />   
</action-mappings> ***Controller Configuration***  
<controller   
processorClass="org.apache.struts.action.RequestProcessor" />   
<message-resources parameter="mybank.ApplicationResources"/>   
</struts-config> ***Message Resource Definition***

However, consider what happens when an instance of this class is serialized across JVMs. The ObjectInputStream will create a new instance of Gender that has the same value as the original instance. So, if you have many thousands objects that have been de-serialized via RMI you might end up with many thousands of extra instances of Gender. The writeReplace() and readResolve() methods are the hook to solve this problem.

class Singleton implements Serializable

{

    int i;

    double d;

    private static final Singleton obj=new Singleton();

    private Singleton()

    {

        i=23;

        d=45.4;

        System.out.println("Executing constructor");

    }

    public static Singleton getInstance()

    {

        System.out.println("An instance is returned");

        return obj;

    }

    public Object readResolve()

    {

        System.out.println("Executing readResolve");

        return Singleton.getInstance();

    }

    @Override

    public String toString() {

    return "Values are :i="+i+" d="+d;

    }

}

Here are some useful guidelines for implementing the hashCode method correctly.

1.     Store an arbitrary non-zero constant integer value (say 7) in an int variable, called hash.

2.     Involve significant variables of your object in the calculation of the hash code, all the variables that are part of equals comparison should be considered for this. Compute an individual hash code int var\_code for each variable var as follows -

a.      If the variable(var) is byte, char, short or int, then var\_code = (int)var;

b.     If the variable(var) is long, then var\_code = (int)(var ^ (var >>> 32));

c.      If the variable(var) is float, then var\_code = Float.floatToIntBits(var);

d.     If the variable(var) is double, then -  
long bits = Double.doubleToLongBits(var);  
var\_code = (int)(bits ^ (bits >>> 32));

e.      If the variable(var) is boolean, then var\_code = var ? 1 : 0;

f.       If the variable(var) is an object reference, then check if it is null, if yes then var\_code = 0; otherwise invoke the hashCode method recursively on this object reference to get the hash code. This can be simplified and given as -  
var\_code = (null == var ? 0 : var.hashCode());

3.     Combine this individual variable hash code var\_code in the original hash code hash as follows -   
hash = 31 \* hash + var\_code;

4.     Follow these steps for all the significant variables and in the end return the resulting integer hash.

5.     Lastly, review your hashCode method and check if it is returning equal hash codes for equal objects. Also, verify that the hash codes returned for the object are consistently the same for multiple invocations during the same execution.

The guidelines provided here for implementing equals and hashCode methods are merely useful as guidelines, these are not absolute laws or rules. Nevertheless, following them while implementing these two methods will certainly give you correct and consistent results.

compostion and aggregation

Introduction

In this article, we will try to understand three important concepts: association, aggregation, and composition.

We will also try to understand in what kind of scenarios we need them. These three concepts have really confused a lot of developers and in this article, my attempt would be to present the concepts in a simplified manner with some real world examples.

Extracting real world relationships from a requirement

The whole point of OOP is that your code replicates real world objects, thus making your code readable and maintainable. When we say real world, the real world has relationships. Let’s consider the simple requirement listed below:

1.       Manager is an employee of XYZ limited corporation.

2.       Manager uses a swipe card to enter XYZ premises.

3.       Manager has workers who work under him.

4.       Manager has the responsibility of ensuring that the project is successful.

5.       Manager's salary will be judged based on project success.

If you flesh out the above five point requirement, we can easily visualize four relationships:-

         Inheritance

         Aggregation

         Association

         Composition

Let’s understand them one by one.

Requirement 1: The IS A relationship

If you look at the first requirement (Manager is an employee of XYZ limited corporation), it’s a parent child relationship or inheritance relationship. The sentence above specifies that Manager is a type of employee, in other words we will have two classes: parent class Employee, and a child class Manager which will inherit from the Employee class.

**Note**: The scope of this article is only limited to aggregation, association, and composition. We will not discuss inheritance in this article as it is pretty straightforward and I am sure you can get 1000s of articles on the net which will help you in understanding it.

Requirement 2: The Using relationship: Association

Requirement 2 is an interesting requirement (Manager uses a swipe card to enter XYZ premises). In this requirement, the manager object and the swipe card object use each other but they have their own object life time. In other words, they can exist without each other. The most important point in this relationship is that there is no single owner.

The above diagram shows how the SwipeCard class uses the Manager class and the Manager class uses theSwipeCard class. You can also see how we can create objects of the Manager class and SwipeCard class independently and they can have their own object life time.

This relationship is called the “Association” relationship.

Requirement 3: The Using relationship with Parent: Aggregation

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement, theManager object will own Worker objects.

The child Worker objects can not belong to any other object. For instance, a Worker object cannot belong to aSwipeCard object.

But… the Worker object can have its own life time which is completely disconnected from the Manager object. Looking from a different perspective, it means that if the Manager object is deleted, the Worker object does not die.

This relationship is termed as an “Aggregation” relationship.

Requirements 4 and 5: The Death relationship: Composition

The last two requirements are actually logically one. If you read closely, the requirements are as follows:

1.       Manager has the responsibility of ensuring that the project is successful.

2.       Manager's salary will be judged based on project success.

Below is the conclusion from analyzing the above requirements:

1.       Manager and the project objects are dependent on each other.

2.       The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good increments if the project has issues.

Below is how the class formation will look like. You can also see that when I go to create the project object, it needs the manager object.

This relationship is termed as the composition relationship. In this relationship, both objects are heavily dependent on each other. In other words, if one goes for garbage collection the other also has to be garbage collected, or putting from a different perspective, the lifetime of the objects are the same. That’s why I have put in the heading “Death” relationship.

Putting things together

Below is a visual representation of how the relationships have emerged from the requirements.

[**1. Dependency Injection**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#dependencyinjection)

[**2. Spring Overview**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#springoverview)

[**3. Spring Installation**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#Installation)

[**4. Datamodel**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#datamodel)

[**5. Using dependency injection with annotations**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#usageannotations)

[**6. Using dependency injection with XML**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#usagexml)

[**7. Summary**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#summary)

[**8. About this website**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#supportandlicense)

[**9. Links and Literature**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#resources)

[**9.1. Source Code**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#sourcecode)

[**9.2. Spring Links**](http://www.vogella.com/tutorials/SpringDependencyInjection/article.html#springlinks)

## 1. Dependency Injection

Java components / classes should be as independent as possible of other Java classes. This increases the possibility to reuse these classes and to test them independently of other classes(Unit Testing). To decouple Java components from other Java components the dependency to a certain other class should get injected into them rather that the class itself creates / finds this object.

A class A has a dependency to class B if class uses class B as a variable.

If dependency injection is used then the class B is given to class A via

* the constructor of the class A - this is then called construction injection
* a setter - this is then called setter injection

The general concept between dependency injection is called Inversion of Control. A class should not configure itself but should be configured from outside.

A design based on independent classes / components increases the re-usability and possibility to test the software. For example, if a class A expects a Dao (Data Access object) for receiving the data from a database you can easily create another test object which mocks the database connection and inject this object into A to test A without having an actual database connection.

A software design based on dependency injection is possible with standard Java.

Spring just simplifies the use of dependency injection by providing a standard way of providing the configuration and by managing the reference to the created objects.

## 2. Spring Overview

The Spring Framework is a very comprehensive framework.

The fundamental functionality provided by the Spring Container is dependency injection. Spring provides a light-weight container, e.g. the Spring core container, for dependency injection (DI).

This container lets you inject required objects into other objects. This results in a design in which the Java class are not hard-coupled. The injection in Spring is either done via setter injection of via construction injection.

These classes which are managed by Spring must conform to the JavaBean standard.

In the context of Spring classes are also referred to as beans or as spring beans.

The Spring core container:

* handles the configuration, generally based on annotations or on an XML file (XMLBeanFactory)
* manages the selected Java classes via the BeanFactory

The core container uses the so-called bean factory to create new objects. New objects are generally created as Singletons if not specified differently.

## 3. Spring Installation

Download Spring from [**http://www.springframework.org/download.**](http://www.springframework.org/download) Select the -with-dependencies.zip to get also all required plugins. At this time of writing I downloaded the version Spring Framework 2.5.5.

The folder "dist" contains the Spring container "spring.jar". The folder lib contains additional require libraries. A minimal Spring application requires the spring.jar, commons-logging.jar (from \lib\jakarta-commons) and log4j\*.jar (from \lib\log4j).

## 4. Datamodel

We will later use the following datamodel for the example.

Create a Java project "de.vogella.spring.di.model" and create the following packages and classes.

**package** writer;

**public** **interface** IWriter {

**public** **void** writer(String s);

}

**package** writer;

**public** **class** Writer **implements** IWriter {

**public** **void** writer (String s){

System.out.println(s);

}

}

**package** writer;

**public** **class** NiceWriter **implements** IWriter {

**public** **void** writer (String s){

System.out.println("The string is " + s);

}

}

**package** testbean;

**import** writer.IWriter;

**public** **class** MySpringBeanWithDependency {

**private** IWriter writer;

**public** **void** setWriter(IWriter writer) {

**this**.writer = writer;

}

**public** **void** run() {

String s = "This is my test";

writer.writer(s);

}

}

The class "MySpringBeanWithDependency.java" contains a setter for the actual writer. We will use the Spring Framework to inject the correct writer into this class.

## 5. Using dependency injection with annotations

As of Spring 2.5 it is possible to configure the dependency injection via annotations. I recommend to use this way of configuring your Spring beans. The next chapter will also describe the way to configure this via XML.

Create a new Java project "de.vogella.spring.di.annotations.first" and include the minimal required spring jars into your classpath.

Copy your model class from the de.vogella.spring.di.model project into this project.

You need now to add annotations to your model to tell Spring which beans should be managed by Spring and how they should be connected.

Add the @Service annotation the MySpringBeanWithDependency.java and NiceWriter.java. Also define with @Autowired on the setWriter method that the property "writer" will be autowired by Spring.

### Tip

@Autowired will tell Spring to search for a Spring bean which implements the required interface and place it automatically into the setter.

**package** testbean;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** writer.IWriter;

*@Service*

**public** **class** MySpringBeanWithDependency {

**private** IWriter writer;

*@Autowired*

**public** **void** setWriter(IWriter writer) {

**this**.writer = writer;

}

**public** **void** run() {

String s = "This is my test";

writer.writer(s);

}

}

**package** writer;

**import** org.springframework.stereotype.Service;

*@Service*

**public** **class** NiceWriter **implements** IWriter {

**public** **void** writer(String s) {

System.out.println("The string is " + s);

}

}

Under the src folder create a folder META-INF and create the following file in this folder. This is the Spring configuration file.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:component-scan base-**package**="testbean" />

<context:component-scan base-**package**="writer" />

</beans>

### Tip

 You can also configure the log4j logger (this is optional) by copying the following file into the source folder.

log4j.rootLogger=FATAL, first

log4j.appender.first=org.apache.log4j.ConsoleAppender

log4j.appender.first.layout=org.apache.log4j.PatternLayout

log4j.appender.first.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

Afer this setup you can wire the application together. Create a main class which reads the configuration file and starts the application.

**package** main;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** testbean.MySpringBeanWithDependency;

**public** **class** Main {

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

ApplicationContext context = **new** ClassPathXmlApplicationContext("META-INF/beans.xml");

BeanFactory factory = context;

MySpringBeanWithDependency test = (MySpringBeanWithDependency) factory

.getBean("mySpringBeanWithDependency");

test.run();

}

}

If you run the application then the class for the IWriterInterface will be inserted into the Test class. By applying the dependency injecting I can later replace this writer with a more sophisticated writer.

As a result the class Test does not depend on the concrete Writer class, is extensible and can be easily tested.

## 6. Using dependency injection with XML

The following example will demonstrate the usage of the dependency injection via xml. The example will inject a writer into another class.

### Tip

 I think annotations rock in general, therefore I recommend not to use the XML configuration but the annotation one. If you have good reason to use the XML configuration please feel free to do so.

Create a new Java project "de.vogella.spring.di.xml.first" and include the minimal required spring jars into your classpath.

Copy your model class from the de.vogella.spring.di.model project into this project.

Under the src folder create a folder META-INF and create the following file in this folder. This is the Spring configuration file.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<bean id="writer" **class**="writer.NiceWriter" />

<bean id="mySpringBeanWithDependency" **class**="testbean.MySpringBeanWithDependency">

<property name="writer" ref="writer" />

</bean>

</beans>

Again, you can now wire the application together. Create a main class which reads the configuration file and starts the application.

**package** main;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** testbean.MySpringBeanWithDependency;

**public** **class** Main {

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

ApplicationContext context = **new** ClassPathXmlApplicationContext("META-INF/beans.xml");

BeanFactory factory = context;

MySpringBeanWithDependency test = (MySpringBeanWithDependency) factory

.getBean("mySpringBeanWithDependency");

test.run();

}

}

# Spring Dependency Injection

JULY 2, 2016 BY [PANKAJ](http://www.journaldev.com/author/pankaj) [34 COMMENTS](http://www.journaldev.com/2410/spring-dependency-injection#comments)

[](http://www.journaldev.com/aspose)

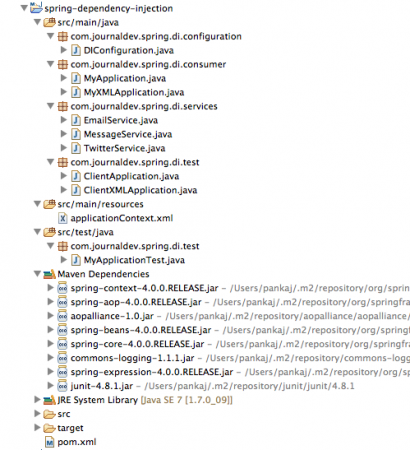
Today we will look into Spring Dependency Injection. [**Spring Framework**](http://www.journaldev.com/2888/spring-tutorial-spring-core-tutorial) core concepts are “**Dependency Injection**” and “**Aspect Oriented Programming**“. I have written earlier about [**Java Dependency Injection**](http://www.journaldev.com/2394/java-dependency-injection-design-pattern-example-tutorial) and how we can use [**Google Guice**](http://www.journaldev.com/2403/google-guice-dependency-injection-example-tutorial) framework to automate this process in our applications.

## Spring Dependency Injection

[](http://cdn.journaldev.com/wp-content/uploads/2013/12/spring-dependency-injection.jpg)

This tutorial is aimed to provide details about Spring Dependency Injection example with both annotation based configuration and XML file based configuration. I will also provide JUnit test case example for the application, since easy testability is one of the major benefits of dependency injection.

I have created spring-dependency-injection maven project whose structure looks like below image.

[](http://cdn.journaldev.com/wp-content/uploads/2013/12/Spring-Dependency-Injection.png)

Let’s look at each of the components one by one.

### Spring Dependency Injection – Maven Dependencies

I have added Spring and JUnit maven dependencies in pom.xml file, final pom.xml code is below.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.journaldev.spring</groupId>

<artifactId>spring-dependency-injection</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.0.0.RELEASE</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

Current stable version of Spring Framework is 4.0.0.RELEASE and JUnit current version is 4.8.1, if you are using any other versions then there might be a small chance that the project will need some change. If you will build the project, you will notice some other jars are also added to maven dependencies because of transitive dependencies, just like above image.

### Spring Dependency Injection – Service Classes

Let’s say we want to send email message and twitter message to the users. For dependency injection, we need to have a base class for the services. So I have MessageService interface with single method declaration for sending message.

package com.journaldev.spring.di.services;

public interface MessageService {

boolean sendMessage(String msg, String rec);

}

Now we will have actual implementation classes to send email and twitter message.

package com.journaldev.spring.di.services;

public class EmailService implements MessageService {

public boolean sendMessage(String msg, String rec) {

System.out.println("Email Sent to "+rec+ " with Message="+msg);

return true;

}

}

package com.journaldev.spring.di.services;

public class TwitterService implements MessageService {

public boolean sendMessage(String msg, String rec) {

System.out.println("Twitter message Sent to "+rec+ " with Message="+msg);

return true;

}

}

Now that our services are ready, we can move on to Component classes that will consume the service.

### Spring Dependency Injection – Component Classes

Let’s write a consumer class for above services. We will have two consumer classes – one with Spring annotations for autowiring and another without annotation and wiring configuration will be provided in the XML configuration file.

package com.journaldev.spring.di.consumer;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.stereotype.Component;

import com.journaldev.spring.di.services.MessageService;

@Component

public class MyApplication {

//field-based dependency injection

//@Autowired

private MessageService service;

// constructor-based dependency injection

// @Autowired

// public MyApplication(MessageService svc){

// this.service=svc;

// }

@Autowired

public void setService(MessageService svc){

this.service=svc;

}

public boolean processMessage(String msg, String rec){

//some magic like validation, logging etc

return this.service.sendMessage(msg, rec);

}

}

Few important points about MyApplication class:

* @Component annotation is added to the class, so that when Spring framework will scan for the components, this class will be treated as component. @Component annotation can be applied only to the class and it’s retention policy is Runtime. If you are not not familiar with Annotations retention policy, I would suggest you to read [java annotations tutorial](http://www.journaldev.com/721/java-annotations-example-tutorial).
* @Autowired annotation is used to let Spring know that autowiring is required. This can be applied to field, constructor and methods. This annotation allows us to implement constructor-based, field-based or method-based dependency injection in our components.
* For our example, I am using method-based dependency injection. You can uncomment the constructor method to switch to constructor based dependency injection.

Now let’s write similar class without annotations.

package com.journaldev.spring.di.consumer;

import com.journaldev.spring.di.services.MessageService;

public class MyXMLApplication {

private MessageService service;

//constructor-based dependency injection

// public MyXMLApplication(MessageService svc) {

// this.service = svc;

// }

//setter-based dependency injection

public void setService(MessageService svc){

this.service=svc;

}

public boolean processMessage(String msg, String rec) {

// some magic like validation, logging etc

return this.service.sendMessage(msg, rec);

}

}

A simple application class consuming the service. For XML based configuration, we can use implement either constructor-based spring dependency injection or method-based spring dependency injection. Note that method-based and setter-based injection approaches are same, it’s just that some prefer calling it setter-based and some call it method-based.

### Spring Dependency Injection Configuration with Annotations

For annotation based configuration, we need to write a Configurator class that will be used to inject the actual implementation bean to the component property.

package com.journaldev.spring.di.configuration;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import com.journaldev.spring.di.services.EmailService;

import com.journaldev.spring.di.services.MessageService;

@Configuration

@ComponentScan(value={"com.journaldev.spring.di.consumer"})

public class DIConfiguration {

@Bean

public MessageService getMessageService(){

return new EmailService();

}

}

Some important points related to above class are:

* @Configuration annotation is used to let Spring know that it’s a Configuration class.
* @ComponentScan annotation is used with @Configuration annotation to specify the packages to look for Component classes.
* @Bean annotation is used to let Spring framework know that this method should be used to get the bean implementation to inject in Component classes.

Let’s write a simple program to test our annotation based Spring Dependency Injection example.

package com.journaldev.spring.di.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.journaldev.spring.di.configuration.DIConfiguration;

import com.journaldev.spring.di.consumer.MyApplication;

public class ClientApplication {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(DIConfiguration.class);

MyApplication app = context.getBean(MyApplication.class);

app.processMessage("Hi Pankaj", "pankaj@abc.com");

//close the context

context.close();

}

}

AnnotationConfigApplicationContext is the implementation of AbstractApplicationContext abstract class and it’s used for autowiring the services to components when annotations are used.AnnotationConfigApplicationContext constructor takes Class as argument that will be used to get the bean implementation to inject in component classes.

getBean(Class) method returns the Component object and uses the configuration for autowiring the objects. Context objects are resource intensive, so we should close them when we are done with it. When we run above program, we get below output.

Dec 16, 2013 11:49:20 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@3067ed13: startup date [Mon Dec 16 23:49:20 PST 2013]; root of context hierarchy

Email Sent to pankaj@abc.com with Message=Hi Pankaj

Dec 16, 2013 11:49:20 PM org.springframework.context.support.AbstractApplicationContext doClose

INFO: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@3067ed13: startup date [Mon Dec 16 23:49:20 PST 2013]; root of context hierarchy

### Spring Dependency Injection XML Based Configuration

We will create Spring configuration file with below data, file name can be anything.

applicationContext.xml code:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">

<!--

<bean id="MyXMLApp" class="com.journaldev.spring.di.consumer.MyXMLApplication">

<constructor-arg>

<bean class="com.journaldev.spring.di.services.TwitterService" />

</constructor-arg>

</bean>

-->

<bean id="twitter" class="com.journaldev.spring.di.services.TwitterService"></bean>

<bean id="MyXMLApp" class="com.journaldev.spring.di.consumer.MyXMLApplication">

<property name="service" ref="twitter"></property>

</bean>

</beans>

Notice that above XML contains configuration for both constructor-based and setter-based spring dependency injection. Since MyXMLApplication is using setter method for injection, the bean configuration contains property element for injection. For constructor based injection, we have to use constructor-argelement.

The configuration XML file is placed in the source directory, so it will be in the classes directory after build.

Let’s see how to use XML based configuration with a simple program.

package com.journaldev.spring.di.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.journaldev.spring.di.consumer.MyXMLApplication;

public class ClientXMLApplication {

public static void main(String[] args) {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(

"applicationContext.xml");

MyXMLApplication app = context.getBean(MyXMLApplication.class);

app.processMessage("Hi Pankaj", "pankaj@abc.com");

// close the context

context.close();

}

}

ClassPathXmlApplicationContext is used to get the ApplicationContext object by providing the configuration files location. It has multiple overloaded constructors and we can provide multiple config files also.

Rest of the code is similar to annotation based configuration test program, the only difference is the way we get the ApplicationContext object based on our configuration choice.

When we run above program, we get following output.

Dec 17, 2013 12:01:23 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@4eeaabad: startup date [Tue Dec 17 00:01:23 PST 2013]; root of context hierarchy

Dec 17, 2013 12:01:23 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [applicationContext.xml]

Twitter message Sent to pankaj@abc.com with Message=Hi Pankaj

Dec 17, 2013 12:01:23 AM org.springframework.context.support.AbstractApplicationContext doClose

INFO: Closing org.springframework.context.support.ClassPathXmlApplicationContext@4eeaabad: startup date [Tue Dec 17 00:01:23 PST 2013]; root of context hierarchy

Notice that some of the output is written by Spring Framework. Since Spring Framework uses log4j for logging purpose and I have not configured it, the output is getting written to console.

### Spring Dependency Injection JUnit Test Case

One of the major benefit of dependency injection in spring is the ease of having mock service classes rather than using actual services. So I have combined all of the learning from above and written everything in a single JUnit 4 test class for dependency injection in spring.

package com.journaldev.spring.di.test;

import org.junit.Assert;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import com.journaldev.spring.di.consumer.MyApplication;

import com.journaldev.spring.di.services.MessageService;

@Configuration

@ComponentScan(value="com.journaldev.spring.di.consumer")

public class MyApplicationTest {

private AnnotationConfigApplicationContext context = null;

@Bean

public MessageService getMessageService() {

return new MessageService(){

public boolean sendMessage(String msg, String rec) {

System.out.println("Mock Service");

return true;

}

};

}

@Before

public void setUp() throws Exception {

context = new AnnotationConfigApplicationContext(MyApplicationTest.class);

}

@After

public void tearDown() throws Exception {

context.close();

}

@Test

public void test() {

MyApplication app = context.getBean(MyApplication.class);

Assert.assertTrue(app.processMessage("Hi Pankaj", "pankaj@abc.com"));

}

}

The class is annotated with @Configuration and @ComponentScan annotation because getMessageService()method returns the MessageService mock implementation. That’s why getMessageService() is annotated with @Bean annotation.

Since I am testing MyApplication class that is configured with annotation, I am usingAnnotationConfigApplicationContext and creating it’s object in the setUp() method. The context is getting closed in tearDown() method. test() method code is just getting the component object from context and testing it.

Do you wonder how Spring Framework does the autowiring and calling the methods that are unknown to Spring Framework. It’s done with the heavy use of [Java Reflection](http://www.journaldev.com/1789/java-reflection-example-tutorial) that we can use to analyze and modify the behaviors of the classes at runtime.

**4. What is difference between List<? extends T>  and  List <? super T> ?**

This is related to previous generics interview questions, some time instead of asking what is bounded and unbounded wildcards interviewer present this question to gauge your understanding of generics. Both of List declaration is e[xample of bounded wildcards,](http://javarevisited.blogspot.sg/2012/04/what-is-bounded-and-unbounded-wildcards.html) List<? extends T> will accept any List with Type extending T while List<? super T> will accept any List with type super class of T. for Example List<? extends Number> can accept List<Integer> orList<Float>. see more on above link.

**5. How to write a generic method which accepts generic argument and return Generic Type?**

writing generic method is not difficult, instead of using raw type you need to use Generic Type like T, E or K,V which are well known placeholders for Type, Element and Key, Value. Look on [Java Collection framework](http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html) for examples of generics methods. In simplest form a generic method would look like:

**public** V put(K key, V value) {  
        **return** cache.put(key, value);  
}

**6. How to write parametrized class in Java using Generics ?**

This is an extension of previous Java generics interview question. Instead of asking to write Generic method Interviewer may ask to *write a type safe class using generics*. again key is instead of using raw types you need to used generic types and always use standard place holder used in JDK.

**7. Write a program to implement LRU cache using Generics ?**

This is an exercise for anyone who like [Coding in Java](http://javarevisited.blogspot.sg/2011/09/code-review-checklist-best-practice.html). One hint is that LinkedHashMap can be used implement fixed size LRU cache  where one needs to remove eldest entry when Cache is full. LinkedHashMap provides a method called removeEldestEntry() which is called by put() and putAll() and can be used to instruct to remove eldest entry. you are free to come up with your own implementation as long as you have a written a working version along with [JUnit test](http://javarevisited.blogspot.sg/2012/06/junit4-annotations-test-examples-and.html).

**8. Can you pass List<String> to a method which accepts List<Object>**

This generic interview question in Java may look confusing to any one who is not very familiar with Generics as in fist glance it looks like String is object so List<String> can be used where List<Object> is required but this is not true. It will result in compilation error. It does make sense if you go one step further because List<Object> can store any any thing including [String, Integer](http://javarevisited.blogspot.com/2011/08/convert-string-to-integer-to-string.html) etc but List<String> can only store Strings.

**List**<**Object**> objectList;  
**List**<**String**> stringList;  
        
objectList = stringList;  *//compilation error incompatible types*

**9. Can we use Generics with Array?**

This was probably most simple generics interview question in Java, if you know the fact that Array doesn't support Generics and that's why Joshua Bloch suggested in Effective Java to prefer List over Array because *List can provide compile time type-safety* over Array.

**10. How can you suppress unchecked warning in Java ?**

javac compiler for Java 5 generates unchecked warnings if you use combine raw types and generics types e.g.

**List**<**String**> rawList = **new** **ArrayList**()  
Note: Hello.java uses unchecked or unsafe operations.;

which can be suppressed by using @SuppressWarnings("unchecked") annotation.

**List**<?> listOfAnyType;   is  **raw type List in Java **can pass any parametrized type to raw type List

**Java Generics Interview questions Update:**

I got few more interview questions on Generics in Java to share with you guys, These questions focus on What is difference between Generics type and Raw type and Can we use Object in place of bounded wildcards etc:

**Difference between List<Object> and raw type List in Java?**

Main difference between raw type and parametrized type List<Object> is that, [compiler](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html)will not check type-safety of raw type at compile time but it will do that for parametrized type and by using Object as Type it inform compiler that it can hold any Type of Object e.g. String or Integer. This Java Generics interview question is based on correct understanding of  raw type in Generics. Any way second difference between them is that you can pass any parametrized type to raw type List but you can not pass List<String> to any method which accept List<Object> it will result in compilation error. Read [How Generics works in Java](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html) for more details.

**Difference between List<?> and List<Object> in Java?**

This generics interview question may look related to previous interview questions but completely different. *List<?> is List of unknown type*while *List<Object> is essentially List of any Type*. You can assign List<String>, List<Integer> to List<?> but you can not assign List<String> to List<Object>.

**List**<?> listOfAnyType;  
**List**<**Object**> listOfObject = **new** **ArrayList**<**Object**>();  
**List**<**String**> listOfString = **new** **ArrayList**<**String**>();  
**List**<**Integer**> listOfInteger = **new** **ArrayList**<**Integer**>();  
        
listOfAnyType = listOfString; *//legal*  
listOfAnyType = listOfInteger; *//legal*  
listOfObjectType = (**List**<**Object**>) listOfString; *//compiler error - in-convertible types*

to know more about wildcards see [Generics Wildcards Examples in Java](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html)

**Difference between List<String> and raw type List.**

This Generics interview question is similar to difference between raw type and parametrized type.Parametrized type are type-safe and type-safety will be guaranteed by compiler but [List raw type is not type safe](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html). You can not store any other Object on List of String but you can not store any Object in raw List. There is no casting required in case of Parametrized type with Generics but explicit casting will be needed for raw type.

**List** listOfRawTypes = **new** **ArrayList**();  
listOfRawTypes.add("abc");  
listOfRawTypes.add(123); *//compiler will allow this - exception at runtime*  
**String** item = (**String**) listOfRawTypes.get(0); *//explicit cast is required*  
item = (**String**) listOfRawTypes.get(1); *//ClassCastException because Integer can not be cast in String*  
        
**List**<**String**> listOfString = **new** **ArrayList**();  
listOfString.add("abcd");  
listOfString.add(1234); *//compiler error, better than runtime Exception*  
item = listOfString.get(0); *//no explicit casting is required - compiler auto cast*

## he answer and explanation to finding the nth highest salary in SQL

|  |
| --- |
|  |

Here we will present one possible answer to finding the nth highest salary first, and the explanation of that answer after since it’s actually easier to understand that way. Note that the first answer we present is actually not optimal from a performance standpoint since it uses a subquery, but we think that it will be interesting for you to learn about because you might just learn something new about SQL. If you want to see the more optimal solutions first, you can skip down to the sections that says “Find the nth highest salary without a subquery” instead.

The SQL below will give you the correct answer – but you will have to plug in an actual value for N of course. This SQL to find the Nth highest salary should work in SQL Server, MySQL, DB2, Oracle, Teradata, and almost any other RDBMS:

SELECT \* /\*This is the outer query part \*/

FROM Employee Emp1

WHERE (N-1) = ( /\* Subquery starts here \*/

SELECT COUNT(DISTINCT(Emp2.Salary))

FROM Employee Emp2

WHERE Emp2.Salary > Emp1.Salary)

## How does the query above work?

The query above can be quite confusing if you have not seen anything like it before – pay special attention to the fact that “Emp1″ appears in both the subquery (also known as an inner query) and the “outer” query. The outer query is just the part of the query that is not the subquery/inner query – both parts of the query are clearly labeled in the comments.

## The subquery is a correlated subquery

The subquery in the SQL above is actually a specific type of subquery known as a***correlated*** subquery. The reason it is called a correlated subquery is because the the subquery uses a value from the outer query in it’s WHERE clause. In this case that value is the Emp1 table alias as we pointed out earlier. A normal subquery can be run***independently*** of the outer query, but a *correlated* subquery can **NOT** be run independently of the outer query. If you want to read more about the differences between correlated and uncorrelated subqueries you can go here: [Correlated vs Uncorrelated Subqueries](http://www.programmerinterview.com/index.php/database-sql/correlated-vs-uncorrelated-subquery/).

You can probably see that Emp1 and Emp2 are just aliases for the same Employee table – it’s like we just created 2 separate clones of the Employee table and gave them different names.

## Understanding and visualizing how the query above works

Let’s assume that we are using this data:

|  |  |
| --- | --- |
| Employee | |
| Employee ID | Salary |
| 3 | 200 |
| 4 | 800 |
| 7 | 450 |

For the sake of our explanation, let’s assume that N is 2 – so the query is trying to find the 2nd highest salary in the Employee table. The first thing that the query above does is process the very first row of the Employee table, which has an alias of Emp1.

The salary in the first row of the Employee table is 200. Because the subquery is correlated to the outer query through the alias Emp1, it means that when the first row is processed, the query will essentially look like this – note that all we did is replace Emp1.Salary with the value of 200:

SELECT \*

FROM Employee Emp1

WHERE (1) = (

SELECT COUNT(DISTINCT(Emp2.Salary))

FROM Employee Emp2

WHERE Emp2.Salary > 200)

So, what exactly is happening when that first row is processed? Well, if you pay special attention to the subquery you will notice that it’s basically searching for the count of salary entries in the Employee table that are greater than 200. Basically, the subquery is trying to find how many salary entries are greater than 200. Then, that count of salary entries is checked to see if it equals 1 in the outer query, and if so then everything from that particular row in Emp1 will be returned.

Note that Emp1 and Emp2 are both aliases for the same table – Employee. Emp2 is only being used in the subquery to compare *all* the salary values to the current salary value chosen in Emp1. This allows us to find the number of salary entries (the count) that are greater than 200. And if this number is equal to N-1 (which is 1 in our case) then we know that we have a winner – and that we have found our answer.

But, it’s clear that the subquery will return a 2 when Emp1.Salary is 200, because there are clearly 2 salaries greater than 200 in the Employee table. And since 2 is not equal to 1, the salary of 200 will clearly not be returned.

So, what happens next? Well, the SQL processor will move on to the next row which is 800, and the resulting query looks like this:

SELECT \*

FROM Employee Emp1

WHERE (1) = (

SELECT COUNT(DISTINCT(Emp2.Salary))

FROM Employee Emp2

WHERE Emp2.Salary > 800)

Since there are no salaries greater than 800, the query will move on to the last row and will of course find the answer as 450. This is because 800 is greater than 450, and the count will be 1. More precisely, the entire row with the desired salary would be returned, and this is what it would look like:

|  |  |
| --- | --- |
| EmployeeID | Salary |
| 7 | 450 |

It’s also worth pointing out that the reason DISTINCT is used in the query above is because there may be duplicate salary values in the table. In that scenario, we only want to count repeated salaries just once, which is exactly why we use the DISTINCT operator.

## A high level summary of how the query works

Let’s go through a high level summary of how someone would have come up with the SQL in the first place – since we showed you the answer first without really going through the thought process one would use to arrive at that answer.

Think of it this way – we are looking for a pattern that will lead us to the answer. One way to look at it is that the 2nd highest salary would have just one salary that is greater than it. The 4th highest salary would have 3 salaries that are greater than it. In more general terms, ***in order to find the Nth highest salary, we just find the salary that has exactly N-1 salaries greater than itself***. And that is exactly what the query above accomplishes – it simply finds the salary that has N-1 salaries greater than itself and returns that value as the answer.

## Find the nth highest salary using the TOP keyword in SQL Server

We can also use the TOP keyword (for databases that support the TOP keyword, like SQL Server) to find the nth highest salary. Here is some fairly simply SQL that would help us do that:

SELECT TOP 1 Salary

FROM (

      SELECT DISTINCT TOP N Salary

      FROM Employee

      ORDER BY Salary DESC

      ) AS Emp

ORDER BY Salary

To understand the query above, first look at the subquery, which simply finds the N highest salaries in the Employee table and arranges them in descending order. Then, the outer query will actually rearrange those values in ascending order, which is what the very last line “ORDER BY Salary” does, because of the fact that the [ORDER BY Default](http://www.programmerinterview.com/index.php/database-sql/sql-order-by-default-sort-order/) is to sort values in ASCENDING order. Finally, that means the Nth highest salary will be at the top of the list of salaries, which means we just want the first row, which is exactly what “SELECT TOP 1 Salary” will do for us!

## Find the nth highest salary without using the TOP keyword

There are many other solutions to finding the nth highest salary that do not need to use the TOP keyword, one of which we already went over. Keep reading for more solutions.

## Find the nth highest salary in SQL without a subquery

The solution we gave above actually does *not* do well from a performance standpoint. This is because the use of the subquery can really slow down the query. With that in mind, let’s go through some different solutions to this problem for different database vendors. Because each database vendor (whether it’s MySQL, Oracle, or SQL Server) has a different SQL syntax and functions, we will go through solutions for specific vendors. But keep in mind that the solution presented above using a subquery should work across different database vendors.

## Find the nth highest salary in MySQL

In MySQL, we can just use the LIMIT clause along with an offset to find the nth highest salary. If that doesn’t make sense take a look at the MySQL-specific SQL to see how we can do this:

SELECT Salary FROM Employee

ORDER BY Salary DESC LIMIT n-1,1

Note that the DESC used in the query above simply arranges the salaries in descending order – so from highest salary to lowest. Then, the key part of the query to pay attention to is the “LIMIT N-1, 1″. The LIMIT clause takes two arguments in that query – the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. So, it’s saying that the offset of the first row to return should be N-1, and the max number of rows to return is 1. What exactly is the offset? Well, the offset is just a numerical value that represents the number of rows from the very first row, and since the rows are arranged in descending order we know that the row at an offset of N-1 will contain the (N-1)th highest salary.

## Find the nth highest salary in SQL Server

In SQL Server, there is no such thing as a LIMIT clause. But, we can still use the offset to find the nth highest salary without using a subquery – just like the solution we gave above in MySQL syntax. But, the SQL Server syntax will be a bit different. Here is what it would look like:

SELECT Salary FROM Employee

ORDER BY Salary DESC OFFSET N-1 ROW(S)

FETCH FIRST ROW ONLY

Note that I haven’t personally tested the SQL above, and I believe that it will only work in SQL Server 2012 and up. Let me know in the comments if you notice anything else about the query.

## Find the nth highest salary in Oracle using rownum

Oracle syntax doesn’t support using an offset like MySQL and SQL Server, but we can actually use the row\_number analytic function in Oracle to solve this problem. Here is what the Oracle-specific SQL would look like to find the nth highest salary:

select \* from (

  select Emp.\*,

row\_number() over (order by Salary DESC) rownumb

from Employee Emp

)

where rownumb = n;  /\*n is nth highest salary\*/

The first thing you should notice in the query above is that inside the subquery the salaries are arranged in descending order. Then, the row\_number analytic function is applied against the list of descending salaries. Applying the row\_number function against the list of descending salaries means that each row will be assigned a row number starting from 1. And since the rows are arranged in descending order the row with the highest salary will have a 1 for the row number. Note that the row number is given the alias rownumb in the SQL above.

This means that in order to find the 3rd or 4th highest salary we simply look for the 3rd or 4th row. The query above will then compare the rownumb to n, and if they are equal will return everything in that row. And that will be our answer!

**import** **static** org.junit.Assert.*assertEquals*;

**import** java.util.LinkedList;

**import** java.util.Queue;

**import** java.util.Stack;

**import** org.junit.Test;

**public** **class** BST {

**public** TreeNode root = **null**;

**public** TreeNode get(**int** element){

**if**(root == **null**){

**return** **null**;

}

TreeNode runner = root;

**while** (**true**){

**if**(runner.data > element){

**if**(runner.leftNode == **null**){

**return** **null**;

}

runner = runner.leftNode;

} **else** **if**(runner.data < element) {

**if**(runner.rightNode == **null**){

**return** **null**;

}

runner = runner.rightNode;

} **else** {

**return** runner;

}

}

}

**public** **void** insert(**int** element){

**if**(root == **null**){

root = **new** TreeNode(element);

**return**;

}

TreeNode runner = root;

**while** (runner.data != element){

**if**(runner.data > element){

**if**(runner.leftNode == **null**){

runner.leftNode = **new** TreeNode(element);

**return**;

}

runner = runner.leftNode;

} **else** {

**if**(runner.rightNode == **null**){

runner.rightNode = **new** TreeNode(element);

**return**;

}

runner = runner.rightNode;

}

}

}

**public** **static** **void** breathFirstSearch(TreeNode root){

Queue<TreeNode> queue = **new** LinkedList<TreeNode>();

queue.add(root);

TreeNode runner = root;

**while**(!queue.isEmpty()){

runner = (TreeNode)queue.remove();

**if**(runner.leftNode != **null**){

queue.add(runner.leftNode);

}

**if**(runner.rightNode != **null**){

queue.add(runner.rightNode);

}

System.*out*.println("visited node " + runner.data);

}

}

**public** **static** **void** depthFirstSearch(TreeNode root){

Stack<TreeNode> stack = **new** Stack<TreeNode>();

**if**(root == **null**){

**return**;

}

TreeNode runner = **null**;

stack.add(root);

**while**(!stack.empty()){

runner = stack.peek();

**if**(runner.leftNode != **null** && !runner.leftNode.visited){

stack.add(runner.leftNode);

**continue**;

}

**if**(runner.rightNode != **null** && !runner.rightNode.visited){

stack.add(runner.rightNode);

**continue**;

}

stack.pop();

runner.visited = **true**;

System.*out*.println("visited node " + runner.data);

}

}

**public** **static** **void** preOrderTraversal(TreeNode root){

**if**(root == **null**){

**return**;

}

System.*out*.print(root.data + " ");

*preOrderTraversal*(root.leftNode);

*preOrderTraversal*(root.rightNode);

}

**public** **static** **void** inOrderTraversal(TreeNode root){

**if**(root == **null**){

**return** ;

}

*inOrderTraversal*(root.leftNode);

System.*out*.print(root.data + " ");

*inOrderTraversal*(root.rightNode);

}

**private** **class** TreeNode {

**public** **int** data;

**public** TreeNode leftNode;

**public** TreeNode rightNode;

**public** **boolean** visited;

TreeNode(**int** data){

**this**.data = data;

}

}

@Test

**public** **void** basicTest(){

BST tree = **new** BST();

**int**[] data = {9, 5, 3, 1, 4, 8, 15, 11, 21, 20, 29};

**for** (**int** i : data){

tree.insert(i);

}

*assertEquals*(tree.get(3).data, 3);

}

}===================================================================

The above implicit requirements are called Non-Functional Requirements. In addition to the above, performance should obviously be a crucial non-functional requirement for all public facing websites.

So, with all the above functional requirements we can build the system by decomposing it into various components band taking care of the non-functional requirements through out the components.

|  |  |
| --- | --- |
| 01 | public class OrderService |
| 02 | { |

|  |  |
| --- | --- |
| 03 | private OrderDAO orderDAO; |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public boolean placeOrder(Order order) |
| 06 | { |

|  |  |
| --- | --- |
| 07 | boolean flag = false; |
| 08 | [logger.info](http://logger.info/)("Entered into OrderService.placeOrder(order) method"); |

|  |  |
| --- | --- |
| 09 | try |
| 10 | { |

|  |  |
| --- | --- |
| 11 | flag = orderDAO.saveOrder(order); |
| 12 | } |

|  |  |
| --- | --- |
| 13 | catch(Exception e) |
| 14 | { |

|  |  |
| --- | --- |
| 15 | logger.error("Error occured in OrderService.placeOrder(order) method"); |
| 16 | } |

|  |  |
| --- | --- |
| 17 | [logger.info](http://logger.info/)("Exiting from OrderService.placeOrder(order) method"); |
| 18 | return flag; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 19 | | } | |  |
| 20 | | } | |  |
| 01 | | public class OrderDAO | |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |
| --- | --- |
| 02 | { |
| 03 | public boolean saveOrder(Order order) |

|  |  |
| --- | --- |
| 04 | { |
| 05 | boolean flag = false; |

|  |  |
| --- | --- |
| 06 | [logger.info](http://logger.info/)("Entered into OrderDAO.saveOrder(order) method"); |
| 07 | Connection conn = null; |

|  |  |
| --- | --- |
| 08 | try |
| 09 | { |

|  |  |
| --- | --- |
| 10 | conn = getConnection();//get database connection |
| 11 | conn.setAutoCommit(false); |

|  |  |
| --- | --- |
| 12 | // insert data into orders\_master table which generates an order\_id |
| 13 | // insert order details into order\_details table with the generated order\_id |

|  |  |
| --- | --- |
| 14 | // insert shipment details into order\_shipment table |
| 15 | conn.commit(); |

|  |  |
| --- | --- |
| 16 | conn.setAutoCommit(true); |
| 17 | flag = true; |

|  |  |
| --- | --- |
| 18 | } |
| 19 | catch(Exception e) |

|  |  |
| --- | --- |
| 20 | { |
| 21 | logger.error("Error occured in OrderDAO.saveOrder(order) method"); |

|  |  |
| --- | --- |
| 22 | conn.rollback(); |
| 23 | } |

|  |  |
| --- | --- |
| 24 | [logger.info](http://logger.info/)("Exiting from OrderDAO.saveOrder(order) method"); |
| 25 | return flag; |

|  |  |
| --- | --- |
| 26 | } |
| 27 | } |

In the code above , the functional requirement implementation and non-functional requirement implementation are mingled in the same place. Logging is placed across the OrderService and OrderDAO classes. At the same time, transaction management is spanned across DAOs.

With this approach we have several issues:

1.    The classes need to be changed either to change functional or non-functional requirements. For example: At some point later in the development if the Team decides to log the Method Entry/Exit information along with TimeStamp, we need to change almost all the classes.

2.    The Transaction Management code setting the auto-commit to false in the beginning, doing the DB operations, committing/rollbacking the operation logic will be duplicated across all the DAOs.

This kind of requirements which span across the modules/components is referred to as Cross Cutting Concerns. To better design the system we should separate these cross cutting concerns from the actual business logic so that it will be easier to change or enhance or maintain the application at a later point.

Aspect Oriented Programming is a methodology which enables the separation of the cross cutting concerns from the actual business logic. So, let us follow the AOP methodology and redesign the above two classes separating the cross cutting concerns.

|  |  |
| --- | --- |
| 1 | public interface IOrderService |
| 2 | { |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | | public boolean placeOrder(Order order); | |  |
| 4 | | } | |  |
| 1 | | public class OrderService implements IOrderService | |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |
| --- | --- |
| 2 | { |
| 3 | private OrderDAO orderDAO; |

|  |  |
| --- | --- |
| 4 |  |
| 5 | public boolean placeOrder(Order order) |

|  |  |
| --- | --- |
| 6 | { |
| 7 | return orderDAO.saveOrder(order); |

|  |  |
| --- | --- |
| 8 | } |
| 9 | } |

|  |  |
| --- | --- |
| 01 | public class OrderDAO |
| 02 | { |

|  |  |
| --- | --- |
| 03 | public boolean saveOrder(Order order) |
| 04 | { |

|  |  |
| --- | --- |
| 05 | boolean flag =false; |
| 06 |  |

|  |  |
| --- | --- |
| 07 | Connectoin conn = null; |
| 08 | try |

|  |  |
| --- | --- |
| 09 | { |
| 10 | conn = getConnection();//get database connection |

|  |  |
| --- | --- |
| 11 | // insert data into orders\_master table which generates an order\_id |
| 12 | // insert order details into order\_details table with the generated order\_id |

|  |  |
| --- | --- |
| 13 | // insert shipment details into order\_shipment table |
| 14 | flag = true; |

|  |  |
| --- | --- |
| 15 | } |
| 16 | catch(Exception e) |

|  |  |
| --- | --- |
| 17 | { |
| 18 | logger.error(e); |

|  |  |
| --- | --- |
| 19 | } |
| 20 | return flag; |

|  |  |
| --- | --- |
| 21 | } |
| 22 | } |

Now lets create a LoggingInterceptor implementing how logging should be done and create a Proxy for OrderService which takes the call from the caller, logs the entry/exit entries using LoggingInterceptor and finally delegates to the actual OrderService.

By using Dynamic Proxies we can separate out the implementation of the cross cutting concerns (such as logging) from actual business logic as follows:

|  |  |
| --- | --- |
| 01 | public class LoggingInterceptor |
| 02 | { |

|  |  |
| --- | --- |
| 03 | public void logEntry(Method m) |
| 04 | { |

|  |  |
| --- | --- |
| 05 | [logger.info](http://logger.info/)("Entered into "+m.getName()+" method"); |
| 06 | } |

|  |  |
| --- | --- |
| 07 | public void logExit(Method m) |
| 08 | { |

|  |  |
| --- | --- |
| 09 | [logger.info](http://logger.info/)("Exiting from "+m.getName()+" method"); |
| 10 | } |

**===================================================**

**Lower Bounded Wildcards**

The [Upper Bounded Wildcards](https://docs.oracle.com/javase/tutorial/java/generics/upperBounded.html) section shows that an upper bounded wildcard restricts the unknown type to be a specific type or a subtype of that type and is represented using the extends keyword. In a similar way, a *lower bounded*wildcard restricts the unknown type to be a specific type or a *super type* of that type.

A lower bounded wildcard is expressed using the wildcard character ('?'), following by the super keyword, followed by its *lower bound*: <? super A>.

**Note:** You can specify an upper bound for a wildcard, or you can specify a lower bound, but you cannot specify both.

Say you want to write a method that puts Integer objects into a list. To maximize flexibility, you would like the method to work on List<Integer>, List<Number>, and List<Object> — anything that can hold Integer values.

To write the method that works on lists of Integer and the supertypes of Integer, such as Integer, Number, and Object, you would specify List<? super Integer>. The term List<Integer> is more restrictive than List<? super Integer> because the former matches a list of type Integer only, whereas the latter matches a list of any type that is a supertype of Integer.

The following code adds the numbers 1 through 10 to the end of a list:

public static void addNumbers(List<? super Integer> list) {

    for (int i = 1; i <= 10; i++) {

        list.add(i);

    }

}

So, after having read the many answers to the question, here is my understanding:

1.    ? extends T means **any class which extends T**. Thus, we are referring to the *children of T*. Hence, **T is the upper bound. The upper-most class in the inheritance hierarchy**

2.    ? super T means **any class / interface which is super of T**. Thus we are referring to all the*parents of T*. **T is thus the lower bound. The lower-most class in the inheritance hierarch**

To write the method that works on lists of Number and the subtypes of Number, such as Integer, Double, and Float, you would specify List<? extends Number>. The term List<Number> is more restrictive than List<? extends Number> because the former matches a list of type Number only, whereas the latter matches a list of type Number or any of its subclasses.

Consider the following process method:

public static void process(List**<? extends Foo>** list) { /\* ... \*/ }

Codility Java Solutions – MissingInteger

         [Codility](http://tarunbajaj.xyz/category/codility/)

         [Java](http://tarunbajaj.xyz/category/java/)

*by*[*tarun*](http://tarunbajaj.xyz/author/admin/)*- November 17, 2015*1

Problem Name : [MissingInteger](https://codility.com/programmers/task/missing_integer)

Difficulty : **Painless**

Assumptions :

         N is an integer within the range [1..100,000];

         each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].

Complexity :

         expected worst-case time complexity is O(N);

         expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Solution :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public int solution (int[] A)  {  int n = A.length;  Arrays.sort(A);  int start = Arrays.binarySearch(A,1);    if(start > -1)  {    for(int i = start + 1; i < n; i++)    {     if(A[i] > A[i-1] + 1)      return A[i-1] + 1;    }      return A[n-1] + 1;  }  else  return 1;  } |

Given 2D Array of only 0s and 1s, find the row which gives maxdecimal value : <http://www.careercup.com/question?id=5735069190717440>

public static int getRowWithMaximumDecimal(int[][] twoDarray){

                 if(twoDarray.length==0)

                          throw new IllegalArgumentException("Invalid Array");

                 int max=0;

                 int maxRow = 0;

                 for(int i=0; i<twoDarray.length;i++){

                          int[] row = twoDarray[i];

                          int dec=0;

                          System.out.println("Computing Decimal value for Row "+i);

                          for(int j=0; j<row.length; j++){

                                   dec = (int) (dec+(row[j]\*Math.pow(2, (row.length-1)-j)));

                          }

                          System.out.println("Dec: "+dec);

                          if(dec>max){

                                   max=dec;

                                   maxRow=i;

                          }

                 }

                 return maxRow;

1) Runnable interface is older than Callable, there from JDK 1.0, while Callable is added on Java 5.0.

2) Runnable interface has run() method to define task while Callable interface uses call() method for task definition.

3) run() method does not return any value, it's return type is void while call method returns value. Callable interface is a [generic parameterized interface](http://javarevisited.blogspot.sg/2012/08/how-to-write-parametrized-class-method-Generic-example.html) and Type of value is provided, when instance of Callable implementation is created.

4) Another difference on run and call method is that run method can not [throw](http://java67.blogspot.sg/2012/10/difference-between-throw-vs-throws-in.html) checked exception, while call method can throw checked exception in Java.

Read more: <http://java67.blogspot.com/2013/01/difference-between-callable-and-runnable-java.html#ixzz3odMz0vUD>

===================

The Runnable interface has been around since the beginning of time for the Java platform. It allows you to define a task to be completed by a thread. As most people probably know already, it offers a single method run() that accepts no arguments and returns no values, nor can it throw any checked exceptions. If you need to get a value back from the now-completed task, you must use a method outside the interface and wait for some kind of notification message that the task completed. For example, the following demonstrates what you might do for just such a scenario:

  Runnable runnable = ...;

  Thread t = new Thread(runnable);

  t.start();

  t.join();

  String value = someMethodtoGetSavedValue()

Nothing is inherently wrong with this code, but it can be done differently now, thanks to the Callable interface introduced in J2SE 5.0. Instead of having a run() method, the Callable interface offers a call() method, which can return an Object or, more specifically, any type that is introduced in the genericized form:

  public interface Callable<V> {

     V call() throws Exception;

  }

Because you cannot pass a Callable into a Thread to execute, you instead use the ExecutorService to execute theCallable object. The service accepts Callable objects to run by way of the submit() method:

  <T> Future<T> submit(Callable<T> task)

As the method definition shows, submitting a Callable object to the ExecutorService returns a Future object. Theget() method of Future will then block until the task is completed. This is the equivalent of the join() call in the first example. Actually, it is the equivalent of both the join() call and the get value call as get() returns the value calculated by the Callable instance.

To demonstrate, the following example creates separate Callable instances for each word passed in on the command line and sums up their length. Each Callable will just calculate the sum of its individual word. The set ofFuture objects are saved to acquire the calculated value from each. If the order of the returned values needed to be preserved, a List could be used instead.

import java.util.\\*;

import java.util.concurrent.\\*;

public class CallableExample {

  public static class WordLengthCallable

        implements Callable {

    private String word;

    public WordLengthCallable(String word) {

      this.word = word;

    }

    public Integer call() {

      return Integer.valueOf(word.length());

    }

  }

  public static void main(String args[]) throws Exception {

    ExecutorService pool = Executors.newFixedThreadPool(3);

    Set<Future<Integer>> set = new HashSet<Future≶Integer>>();

    for (String word: args) {

      Callable<Integer> callable = new WordLengthCallable(word);

      Future<Integer> future = pool.submit(callable);

      set.add(future);

    }

    int sum = 0;

    for (Future<Integer> future : set) {

      sum += future.get();

    }

    System.out.printf("The sum of lengths is %s%n", sum);

    System.exit(sum);

  }

}

The WordLengthCallable saves each word and uses the word's length as the value returned by the call() method. This value could take some time to generate but in this case is known immediately. The only requirement of call() is the value is returned at the end of the call. When the get() method of Future is later called, the Future will either have the value immediately if the task runs quickly, as in this case, or will wait until the value is done generating. Multiple calls to get() will not cause the task to be rerun in the thread.

Because the goal of the program is to calculate the sum of all word lengths, it doesn't matter in which order theCallable tasks finish. It is perfectly OK if the last task completes before the first three. The first get() call to Futurewill just wait for the first task in the Set to complete. This does not block other tasks from running to completion separately. It is just waiting for that one thread or task to complete.

This particular example uses a fixed-size thread pool for the ExecutorService, but any available service will do.

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed. A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.**java.util.concurrent.Executors** provide implementation of**java.util.concurrent.Executor** interface to create the thread pool in java.  
Let’s write a simple program to explain it’s working.

First we need to have a Runnable class.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | public class WorkerThread implements Runnable { |
| 04 |  |

|  |  |
| --- | --- |
| 05 | private String command; |
| 06 |  |

|  |  |
| --- | --- |
| 07 | public WorkerThread(String s){ |
| 08 | this.command=s; |

|  |  |
| --- | --- |
| 09 | } |
| 10 |  |

|  |  |
| --- | --- |
| 11 | @Override |
| 12 | public void run() { |

|  |  |
| --- | --- |
| 13 | System.out.println(Thread.currentThread().getName()+' Start. Command = '+command); |
| 14 | processCommand(); |

|  |  |
| --- | --- |
| 15 | System.out.println(Thread.currentThread().getName()+' End.'); |
| 16 | } |

|  |  |
| --- | --- |
| 17 |  |
| 18 | private void processCommand() { |

|  |  |
| --- | --- |
| 19 | try { |
| 20 | Thread.sleep(5000); |

|  |  |
| --- | --- |
| 21 | } catch (InterruptedException e) { |
| 22 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 23 | } |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | @Override |

|  |  |
| --- | --- |
| 27 | public String toString(){ |
| 28 | return this.command; |

|  |  |
| --- | --- |
| 29 | } |
| 30 | } |

Here is the test program where we are creating fixed thread pool from Executors framework.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.concurrent.ExecutorService; |
| 04 | import java.util.concurrent.Executors; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class SimpleThreadPool { |

|  |  |
| --- | --- |
| 07 |  |
| 08 | public static void main(String[] args) { |

|  |  |
| --- | --- |
| 09 | ExecutorService executor = Executors.newFixedThreadPool(5); |
| 10 | for (int i = 0; i < 10; i++) { |

|  |  |
| --- | --- |
| 11 | Runnable worker = new WorkerThread('' + i); |
| 12 | executor.execute(worker); |

|  |  |
| --- | --- |
| 13 | } |
| 14 | executor.shutdown(); |

|  |  |
| --- | --- |
| 15 | while (!executor.isTerminated()) { |
| 16 | } |

|  |  |
| --- | --- |
| 17 | System.out.println('Finished all threads'); |
| 18 | } |

|  |  |
| --- | --- |
| 19 |  |
| 20 | } |

In above program, we are creating fixed size thread pool of 5 worker threads. Then we are submitting 10 jobs to this pool, since the pool size is 5, it will start working on 5 jobs and other jobs will be in wait state, as soon as one of the job is finished, another job from the wait queue will be picked up by worker thread and get’s executed.

Here is the output of the above program.

|  |  |
| --- | --- |
| 01 | pool-1-thread-2 Start. Command = 1 |
| 02 | pool-1-thread-4 Start. Command = 3 |

|  |  |
| --- | --- |
| 03 | pool-1-thread-1 Start. Command = 0 |
| 04 | pool-1-thread-3 Start. Command = 2 |

|  |  |
| --- | --- |
| 05 | pool-1-thread-5 Start. Command = 4 |
| 06 | pool-1-thread-4 End. |

|  |  |
| --- | --- |
| 07 | pool-1-thread-5 End. |
| 08 | pool-1-thread-1 End. |

|  |  |
| --- | --- |
| 09 | pool-1-thread-3 End. |
| 10 | pool-1-thread-3 Start. Command = 8 |

|  |  |
| --- | --- |
| 11 | pool-1-thread-2 End. |
| 12 | pool-1-thread-2 Start. Command = 9 |

|  |  |
| --- | --- |
| 13 | pool-1-thread-1 Start. Command = 7 |
| 14 | pool-1-thread-5 Start. Command = 6 |

|  |  |
| --- | --- |
| 15 | pool-1-thread-4 Start. Command = 5 |
| 16 | pool-1-thread-2 End. |

|  |  |
| --- | --- |
| 17 | pool-1-thread-4 End. |
| 18 | pool-1-thread-3 End. |

|  |  |
| --- | --- |
| 19 | pool-1-thread-5 End. |
| 20 | pool-1-thread-1 End. |

|  |  |
| --- | --- |
| 21 | Finished all threads |

The output confirms that there are five threads in the pool named from “pool-1-thread-1? to “pool-1-thread-5? and they are responsible to execute the submitted tasks to the pool.

**Executors** class provide simple implementation of **ExecutorService** using **ThreadPoolExecutor** but ThreadPoolExecutor provides much more feature than that. We can specify the number of threads that will be alive when we create ThreadPoolExecutor instance and we can limit the size of thread pool and create our own **RejectedExecutionHandler**implementation to handle the jobs that can’t fit in the worker queue.

Here is our custom implementation of RejectedExecutionHandler interface.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

**for**(**int** j=0;j< groupItems.length();j++)

                      {

                      String groupItem=groupItems.get(j).toString();

**if**(!(channelNames).contains(groupItem.toUpperCase()))

                    //  if(!(channelNames).(groupItem.toUpperCase()))

                      {

                            //isAllGroupItemsinChannelList=false;

                             targetGroupItems.put(groupItem);

                      }

                      }

**if**(targetGroupItems.length()>=1){ // case when all group items matches with channel list remove all group items and flow view shapes

                    System.*out*.println("when targetGroupItems.length()>=1 ");

                      JSONObject targetFlowView = **new** JSONObject(flowView.toString());

                      targetFlowView.put("groupItems", targetGroupItems);

**for** (**int** x=0;x<targetGroupItems.length();x++){

                             System.*out*.println("targetGroupItems="+x+" =="+targetGroupItems.get(x));

                      }

**for** (**int** j = 0; j < flowViewShapes.length(); j++) {

                            JSONObject flowViewShape = (JSONObject) flowViewShapes.get(j);

                            JSONObject targetFlowViewShape = **new** JSONObject(flowViewShape.toString());

**49) How is Hibernate accessed using the Spring framework?**

Hibernate can be accessed in the following two ways:

           By IOC with a Callback and HibernateTemplate.

           By applying an AOP Interceptor and broadening the HibernateDaoSupport.

**54) What are all the ways to access Hibernate by using Spring?**

There are two ways to access hibernate using spring:

          Inversion of Control with a Hibernate Template and Callback

          Extending HibernateDAOSupport and Applying an AOP Interceptor node.

**55) How struts application can be integrated with spring?**

There are two options for struts application that can be integrated with spring:

Configuration of Spring to manage beans using ContextLoader plugin and set their dependencies in a spring context file

Grab spring managed beans explicitly using agetwebapplicationcontext()

**56) What is Inversion of control (IOC)?**

Inversion of Control (IOC) is also called as dependency Injection which is nothingbut a design pattern that gives control to the assembler of classes. In general, class will instantiate another class if required.

But in this design pattern, this control has been to given to assembler and assembler will instantiate required class if needed.

**57) Write the benefits of using IOC?**

The major benefits of dependency injection or IOC are that it reduces the amount of coding required for the application. This allows the testing of the application to be done quickly and easily as no JNDI lookup mechanism or singletons are required. IOC containers also support lazy loading and eager installation of services.

# How to call stored procedure in Hibernate

By [mkyong](http://www.mkyong.com/author/mkyong/) | February 15, 2010 | Updated : August 30, 2012

In this tutorial, you will learn how to call a store procedure in Hibernate.

## **MySQL store procedure**

Here’s a MySQL store procedure, which accept a stock code parameter and return the related stock data.

DELIMITER $$

CREATE PROCEDURE `GetStocks`(int\_stockcode varchar(20))

BEGIN

   SELECT \* FROM stock where stock\_code = int\_stockcode;

   END $$

DELIMITER ;

In MySQL, you can simple call it with a **call** keyword :

CALL GetStocks('7277');

## **Hibernate call store procedure**

In Hibernate, there are three approaches to call a database store procedure.

## **1. Native SQL – createSQLQuery**

You can use **createSQLQuery()** to call a store procedure directly.

Query query = session.createSQLQuery(

         "CALL GetStocks(:stockCode)")

         .addEntity(Stock.class)

         .setParameter("stockCode", "7277");

List result = query.list();

for(int i=0; i<result.size(); i++){

         Stock stock = (Stock)result.get(i);

         System.out.println(stock.getStockCode());

}

## **2. NamedNativeQuery in annotation**

Declare your store procedure inside the **@NamedNativeQueries** annotation.

//Stock.java

...

@NamedNativeQueries({

         @NamedNativeQuery(

         name = "callStockStoreProcedure",

         query = "CALL GetStocks(:stockCode)",

         resultClass = Stock.class

         )

})

@Entity

@Table(name = "stock")

public class Stock implements java.io.Serializable {

...

Call it with **getNamedQuery()**.

Query query = session.getNamedQuery("callStockStoreProcedure")

         .setParameter("stockCode", "7277");

List result = query.list();

for(int i=0; i<result.size(); i++){

         Stock stock = (Stock)result.get(i);

         System.out.println(stock.getStockCode());

}

## **3. sql-query in XML mapping file**

Declare your store procedure inside the "**sql-query**" tag.

<!-- Stock.hbm.xml -->

...

<hibernate-mapping>

    <class name="com.mkyong.common.Stock" table="stock" ...>

        <id name="stockId" type="java.lang.Integer">

            <column name="STOCK\_ID" />

            <generator class="identity" />

        </id>

        <property name="stockCode" type="string">

            <column name="STOCK\_CODE" length="10" not-null="true" unique="true" />

        </property>

        ...

    </class>

    <sql-query name="callStockStoreProcedure">

         <return alias="stock" class="com.mkyong.common.Stock"/>

         <![CDATA[CALL GetStocks(:stockCode)]]>

    </sql-query>

</hibernate-mapping>

Call it with **getNamedQuery()**.

Query query = session.getNamedQuery("callStockStoreProcedure")

         .setParameter("stockCode", "7277");

List result = query.list();

for(int i=0; i<result.size(); i++){

         Stock stock = (Stock)result.get(i);

         System.out.println(stock.getStockCode());

}

Java Code Examples for javax.jms.ExceptionListener

The following are 20 Jave code examples that show how to use the javax.jms.ExceptionListener class. These examples are extracted from open source projects. You can click to vote up the examples you like. Your votes will be used in an intelligent system to get more and better code examples. Thanks!

Example 1

9

vote

From project activemq, under directory /activemq-core/src/test/java/org/apache/activemq/, in source fileProducerFlowControlSendFailTest.java

@Override protected ConnectionFactory createConnectionFactory() throws Exception {

ActiveMQConnectionFactory connectionFactory=new ActiveMQConnectionFactory(connector.getConnectUri());

connectionFactory.setExceptionListener(new ExceptionListener(){

public void onException( JMSException arg0){

if (arg0 instanceof ResourceAllocationException) {

gotResourceException.set(true);

}

}

}

);

return connectionFactory;

}

Example 2

8

vote

From project activemq, under directory /activemq-pool/src/test/java/org/apache/activemq/pool/, in source filePooledTopicPublisherTest.java

public void testSetGetExceptionListener() throws Exception {

PooledConnectionFactory pcf=new PooledConnectionFactory();

pcf.setConnectionFactory(new ActiveMQConnectionFactory("vm://test"));

connection=(TopicConnection)pcf.createConnection();

ExceptionListener listener=new ExceptionListener(){

public void onException( JMSException exception){

}

}

;

connection.setExceptionListener(listener);

assertEquals(listener,connection.getExceptionListener());

}

Example 3

8

vote

From project aries, under directory /transaction/transaction-jms/src/main/java/org/apache/aries/transaction/jms/internal/, in source file ConnectionPool.java

public ConnectionPool(Connection connection,ObjectPoolFactory poolFactory) throws JMSException {

this(connection,new ConcurrentHashMap<SessionKey,SessionPool>(),poolFactory);

connection.setExceptionListener(new ExceptionListener(){

public void onException( JMSException exception){

synchronized (ConnectionPool.this) {

hasFailed=true;

}

}

}

);

}

Example 4

8

vote

From project hornetq, under directory /tests/jms-tests/src/test/java/org/hornetq/jms/tests/, in source fileConnectionTest.java

/\*\*

\* Test ExceptionListener stuff

\*/

public void testExceptionListener() throws Exception {

Connection conn=JMSTestCase.cf.createConnection();

ExceptionListener listener1=new MyExceptionListener();

conn.setExceptionListener(listener1);

ExceptionListener listener2=conn.getExceptionListener();

ProxyAssertSupport.assertNotNull(listener2);

ProxyAssertSupport.assertEquals(listener1,listener2);

conn.close();

}

Example 5

7

vote

From project activemq, under directory /activemq-core/src/test/java/org/apache/activemq/, in source fileActiveMQConnectionFactoryTest.java

public void testSetExceptionListener() throws Exception {

ActiveMQConnectionFactory cf=new ActiveMQConnectionFactory("vm://localhost?broker.persistent=false");

connection=(ActiveMQConnection)cf.createConnection();

assertNull(connection.getExceptionListener());

ExceptionListener exListener=new ExceptionListener(){

public void onException( JMSException arg0){

}

}

;

cf.setExceptionListener(exListener);

connection=(ActiveMQConnection)cf.createConnection();

assertNotNull(connection.getExceptionListener());

assertEquals(exListener,connection.getExceptionListener());

connection=(ActiveMQConnection)cf.createConnection();

assertEquals(exListener,connection.getExceptionListener());

assertEquals(exListener,cf.getExceptionListener());

}

Example 6

7

vote

From project activemq, under directory /activemq-core/src/test/java/org/apache/activemq/broker/, in source fileNioQueueSubscriptionTest.java

public void testLotsOfConcurrentConnections() throws Exception {

ExecutorService executor=Executors.newCachedThreadPool();

final ConnectionFactory factory=createConnectionFactory();

final ExceptionListener listener=this;

int connectionCount=400;

for (int i=0; i < connectionCount; i++) {

executor.execute(new Runnable(){

public void run(){

try {

ActiveMQConnection connection=(ActiveMQConnection)factory.createConnection();

connection.setExceptionListener(listener);

connection.start();

assertNotNull(connection.getBrokerName());

connections.add(connection);

}

catch ( Exception e) {

exceptions.put(Thread.currentThread(),e);

}

}

}

);

}

executor.shutdown();

executor.awaitTermination(30,TimeUnit.SECONDS);

if (!exceptions.isEmpty()) {

LOG.error("" + exceptions.size() + " exceptions like",exceptions.values().iterator().next());

fail("unexpected exceptions in worker threads: " + exceptions.values().iterator().next());

}

LOG.info("created " + connectionCount + " connecitons");

}

Example 7

7

vote

From project hornetq, under directory /hornetq-jms/src/main/java/org/hornetq/jms/client/, in source fileHornetQConnection.java

public synchronized void connectionFailed(final HornetQException me,boolean failedOver){

if (me == null) {

return;

}

HornetQConnection conn=connectionRef.get();

if (conn != null) {

try {

final ExceptionListener exceptionListener=conn.getExceptionListener();

if (exceptionListener != null) {

final JMSException je=new JMSException(me.toString(),failedOver ? EXCEPTION\_FAILOVER : EXCEPTION\_DISCONNECT);

je.initCause(me);

new Thread(new Runnable(){

public void run(){

exceptionListener.onException(je);

}

}

).start();

}

}

catch ( JMSException e) {

if (!conn.closed) {

HornetQJMSLogger.LOGGER.errorCallingExcListener(e);

}

}

}

}

Example 8

7

vote

From project hornetq, under directory /tests/integration-tests/src/test/java/org/hornetq/tests/integration/client/, in source fileFailureDeadlockTest.java

public void testDeadlock() throws Exception {

for (int i=0; i < 100; i++) {

final Connection conn1=cf1.createConnection();

Session sess1=conn1.createSession(false,Session.AUTO\_ACKNOWLEDGE);

RemotingConnection rc1=((ClientSessionInternal)((HornetQSession)sess1).getCoreSession()).getConnection();

final Connection conn2=cf2.createConnection();

Session sess2=conn2.createSession(false,Session.AUTO\_ACKNOWLEDGE);

RemotingConnection rc2=((ClientSessionInternal)((HornetQSession)sess2).getCoreSession()).getConnection();

ExceptionListener listener1=new ExceptionListener(){

public void onException( final JMSException exception){

try {

conn2.close();

}

catch ( Exception e) {

FailureDeadlockTest.log.error("Failed to close connection2",e);

}

}

}

;

conn1.setExceptionListener(listener1);

conn2.setExceptionListener(listener1);

Failer f1=new Failer(rc1);

Failer f2=new Failer(rc2);

f1.start();

f2.

**Diagnostic Steps**

Recent versions of browsers made security updates in order to address padding and logjam vulnerabilities.  
Since then access to HDM GUI that requires SLL / HTTPS is not possible.  
The only option till now was to downgrade browser version where this is allowed.

**Solution**

**This procedure is valid for Weblogic Server 10.3.5 ( HDM 4.0 / HDM 4.1 )**

Please follow the steps described below:

1.     Upgrade the Demo SSL certificate of weblogic to 2048 bits one if not yet the case using the procedure below:

         Login to the machine that hosts the Admin server as hdm user ( typically hdm or hdmadmin )

         Navigate to $HDM\_HOME/domains/HDMDomain, where $HDM\_HOME is the hdm installation home directory

         Set the environment using command ./setEnv.sh  
Note : DOT SPACE DOT SLASH setEnv.sh

[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] ./setEnv.sh  
[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->]

         Create a temp directory for generated keys and navigate there.  
e.g. /opt/hdm/domains/HDMDomain/gen\_keys

[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] pwd  
/opt/hdm/domains/HDMDomain  
[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] mkdir gen\_keys  
[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] cd gen\_keys

         Create the new certificate  
java -cp $HDM\_HOME/wlserver\_10.3/server/lib/weblogic.jar utils.CertGen -keyfilepass DemoIdentityPassPhrase -certfile democert -keyfile demokey -strength 2048

[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] java -cp /opt/hdm/wlserver\_10.3/server/lib/weblogic.jar utils.CertGen -keyfilepass DemoIdentityPassPhrase -certfile democert -keyfile demokey -strength 2048  
Generating a certificate with common name HDM1 and key strength 2048  
issued by CA with certificate from /opt/hdm/wlserver\_10.3/server/lib/CertGenCA.der file and key from /opt/hdm/wlserver\_10.3/server/lib/CertGenCAKey.der file

         Check if the needed files are created with the ls command

[hdm@HDM1{ /opt/hdm/domains/HDMDomain/gen\_keys }->] ls -lrt  
total 16  
-rw-rw-r--. 1 hdm hdm 1260 Sep 18 17:25 demokey.der  
-rw-rw-r--. 1 hdm hdm 1781 Sep 18 17:25 demokey.pem  
-rw-rw-r--. 1 hdm hdm  689 Sep 18 17:25 democert.der  
-rw-rw-r--. 1 hdm hdm  989 Sep 18 17:25 democert.pem

         Import the newly created certificate in a new keystore  
java -cp $HDM\_HOME/wlserver\_10.3/server/lib/weblogic.jar utils.ImportPrivateKey -keystore DemoIdentity.jks -storepass DemoIdentityKeyStorePassPhrase -keyfile demoKey -keyfilepass DemoIdentityPassPhrase -certfile democert.pem -keyfile demokey.pem -alias demoidentity

[hdm@HDM1{ /opt/hdm/domains/HDMDomain/gen\_keys }->] java -cp /opt/hdm/wlserver\_10.3/server/lib/weblogic.jar utils.ImportPrivateKey -keystore DemoIdentity.jks -storepass DemoIdentityKeyStorePassPhrase -keyfile demoKey -keyfilepass DemoIdentityPassPhrase -certfile democert.pem -keyfile demokey.pem -alias demoidentity  
No password was specified for the key entry  
Key file password will be used  
<Sep 18, 2015 5:26:25 PM CEST> <Info> <Security> <BEA-090906> <Changing the default Random Number Generator in RSA CryptoJ from ECDRBG to FIPS186PRNG. To disable this change, specify -Dweblogic.security.allowCryptoJDefaultPRNG=true>

         Stop all the servers (Admin server and all managed servers on all physical hosts)

         Make a backup of original DemoIdentity.jks on each physical machine:  
mv $HDM\_HOME/wlserver\_10.3/server/lib/DemoIdentity.jks $HDM\_HOME/wlserver\_10.3/server/lib/DemoIdentity.jks.backup

[hdm@HDM1{ /opt/hdm/domains/HDMDomain/gen\_keys }->] mv /opt/hdm/wlserver\_10.3/server/lib/DemoIdentity.jks /opt/hdm/wlserver\_10.3/server/lib/DemoIdentity.jks.backup

         Copy the new file to weblogic: cp DemoIdentity.jks $HDM\_HOME/wlserver\_10.3/server/lib/  
In case of multiple physical machines copy that file to other servers as well to $HDM\_HOME/wlserver\_10.3/server/lib/

[hdm@HDM1{ /opt/hdm/domains/HDMDomain/gen\_keys }->] cp DemoIdentity.jks /opt/hdm/wlserver\_10.3/server/lib/  
[hdm@HDM1{ /opt/hdm/domains/HDMDomain/gen\_keys }->]

         Start the HDM

2.     At this stage you might already be able to access admin console via IE.  
If yes, you can verify that the certificate was upgraded well by viewing certificate information in browser:

3.     On all managed servers and Adminserver, enable the JSSE SSL implementation (you have to be in "lock and edit" mode of the console):  
HDMDomain -> Environment -> Servers -> Configuration -> SSL -> Advanced -> Check use JSSE SSL  
Save and activate the changes.  
For this action you may need an "older browser"

4.     Shutdown HDM completely

5.     On the server that runs admin server , update config.xml file located under <HDM\_HOME>/domains/HDMDomain/config, for all managed servers and Admin server, by adding two cipher suites in the ssl section of the server just after the <enabled> tag (do not forget to make a backup of config.file before proceeding):

[hdm@HDM1{ /opt/hdm/domains/HDMDomain/config }->] cp config.xml config.xml.bkp

**To be added:**  
<ciphersuite>TLS\_RSA\_WITH\_RC4\_128\_SHA</ciphersuite>  
<ciphersuite>TLS\_RSA\_WITH\_RC4\_128\_MD5</ciphersuite>

The server's SSL section will look like below:

<ssl>  
      <name>AdminServer\_9002</name>  
      <enabled>true</enabled>  
      <ciphersuite>TLS\_RSA\_WITH\_RC4\_128\_SHA</ciphersuite>  
      <ciphersuite>TLS\_RSA\_WITH\_RC4\_128\_MD5</ciphersuite>  
      <hostname-verifier xsi:nil="true"></hostname-verifier>  
      <hostname-verification-ignored>true</hostname-verification-ignored>  
      <export-key-lifespan>500</export-key-lifespan>  
      <client-certificate-enforced>false</client-certificate-enforced>  
      <two-way-ssl-enabled>false</two-way-ssl-enabled>  
      <ssl-rejection-logging-enabled>true</ssl-rejection-logging-enabled>  
      <inbound-certificate-validation>BuiltinSSLValidationOnly</inbound-certificate-validation>  
      <outbound-certificate-validation>BuiltinSSLValidationOnly</outbound-certificate-validation>  
      <allow-unencrypted-null-cipher>false</allow-unencrypted-null-cipher>  
      <use-server-certs>false</use-server-certs>  
      <jsse-enabled>true</jsse-enabled>  
</ssl>

6.     Have **the customer** download the Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files for the Java(TM) Platform, Standard Edition Development Kit, v6. using following link: <http://www.oracle.com/technetwork/java/javase/downloads/jce-6-download-429243.html>

Please read carefully the README file because it contains important legal information, which should be confirmed by the customer

Extract and put 2 jar files on each host of the setup to /tmp directory.

[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] ls -lrt /tmp/\*.jar  
-rw-rw-r--. 1 hdm hdm 2465 Nov 16  2006 /tmp/US\_export\_policy.jar  
-rw-rw-r--. 1 hdm hdm 2481 Nov 16  2006 /tmp/local\_policy.jar

7.     Navigate to security lib directory of the java version used by HDM.  
By default it's <HDM\_HOME>/jdk1.6.0\_33/jre/lib/security/

[hdm@HDM1{ /opt/hdm/domains/HDMDomain }->] cd /opt/hdm/jdk1.6.0\_33/jre/lib/security/

8.     Make a backup of 2 jar files: US\_export\_policy.jar and local\_policy.jar  
cp US\_export\_policy.jar bkp\_US\_export\_policy.jar  
cp local\_policy.jar bkp\_local\_policy.jar

[hdm@HDM1{ /opt/hdm/jdk1.6.0\_33/jre/lib/security }->] cp US\_export\_policy.jar bkp\_US\_export\_policy.jar  
[hdm@HDM1{ /opt/hdm/jdk1.6.0\_33/jre/lib/security }->] cp local\_policy.jar bkp\_local\_policy.jar

9.     On each machine, replace those files with the ones placed earlier in /tmp folder

cp /tmp/US\_export\_policy.jar .  
cp /tmp/local\_policy.jar .

[hdm@HDM1{ /opt/hdm/jdk1.6.0\_33/jre/lib/security }->] cp /tmp/US\_export\_policy.jar .  
cp: overwrite `./US\_export\_policy.jar'? y  
[hdm@HDM1{ /opt/hdm/jdk1.6.0\_33/jre/lib/security }->] cp /tmp/local\_policy.jar .  
cp: overwrite `./local\_policy.jar'? y

10.   Restart all

**Workaround**

**Cause**

Improved security of the browsers required webservers to adjust their algorithms used.

In order to achieve this HDM

         changes the default SSL implementation from Certicom to JSSE which supports TLS protocol in Weblogic Server 10.3.5

         declares additional cipher suites in domain configuration file

         replaces the java security libraries to support such ciphers.

**Information for Internal Audiences**

**Applicable Products & Releases**

What is an SSL Certificate?

[Live Chat with an SSL Expert to have your questions answered in minutes!](https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate/)

SSL Certificates are small data files that digitally bind a [cryptographic key](https://www.globalsign.com/en/ssl-information-center/choosing-safe-key-sizes/) to an organization’s details. When installed on a web server, it activates the padlock and the https protocol (over port 443) and allows secure connections from a web server to a browser. Typically, [SSL](https://www.globalsign.com/en/ssl-information-center/what-is-ssl/) is used to secure credit card transactions, data transfer and logins, and more recently is becoming the norm when securing browsing of social media sites. SSL Certificates bind together:

         A domain name, server name or hostname.

         An organizational identity (i.e. company name) and location.

An organization needs to install the SSL Certificate onto its web server to initiate secure sessions with browsers. Depending on the [type of SSL Certificate](https://www.globalsign.com/en/ssl-information-center/types-of-ssl-certificate/) applied for, the organization will need to go through differing levels of vetting. Once installed, it is possible to connect to the website over [https://www.domain.com](https://www.domain.com/), as this tells the server to establish a secure connection with the browser. Once a secure connection is established, all web traffic between the web server and the web browser will be secure. Browsers tell visitors a website is SSL secure via several visible trust indicators:

[Extended Validation (EV)](https://www.globalsign.com/en/ssl-information-center/what-is-an-extended-validation-certificate/) SSL Certificates (such as GlobalSign [ExtendedSSL](https://www.globalsign.com/en/ssl/ev-ssl/)):

[Standard SSL Certificates](https://www.globalsign.com/en/ssl-information-center/telling-dv-and-ov-certificates-apart/) (such as GlobalSign [DomainSSL](https://www.globalsign.com/en/ssl/domain-ssl/) and [OrganizationSSL](https://www.globalsign.com/en/ssl/organization-ssl/)) display:

GlobalSign offers a range of SSL Certificates with the strongest 2048 bit encryption and value add features to ensure your website is protected.

[Help me choose the right SSL certificate based on my website's needs.](https://www.globalsign.com/en/ssl/what-kind-of-ssl-should-i-get/)

SSL Certificate Details

To view the details of an SSL Certificate, go to a secure site, click on the padlock and select “View Certificate”. All browsers are slightly different, but the Certificate always contains the same information.

To view the actual contents of the Certificate click the "Details" tab:

Click the "Certificate Path" tab to see which Trusted Root Certificate has been used to issue the SSL Certificate:

Why is the Root Certificate important?

SSL Certificates need to be issued from a trusted Certificate Authority's [Root Certificate](https://www.globalsign.com/en/ssl-information-center/certificate-authority-root/), and preferably by a [2048 bit](https://www.globalsign.com/en/ssl-information-center/choosing-safe-key-sizes/) Certificate that's widely distributed. The Root Certificate must be present on the end user's machine in order for the Certificate to be trusted. If it is not trusted the browser will present untrusted error messages to the end user. In the case of e-commerce, such error messages result in immediate lack of confidence in the website and organizations risk losing confidence and business from the majority of consumers.

Companies like GlobalSign are known as trusted [Certificate Authorities](https://www.globalsign.com/en/ssl-information-center/what-are-certification-authorities-trust-hierarchies/). This is because browser and operating system vendors such as Microsoft, Mozilla, Opera, Blackberry, Java, etc., trust that GlobalSign is a legitimate Certificate Authority and that it can be relied on to issue trustworthy SSL Certificates. The more applications, devices and browsers the Certificate Authority embeds its Root into, the better "recognition" the SSL Certificate can provide.

GlobalSign has, for over 15 years, been operating the GlobalSign Ready program for Root Certificate embedding. This program ensures its in-house engineers from America, Europe and Asia are in constant communication with the application, device and browser vendors to ensure the GlobalSign Root Certificate is present everywhere that may be used for SSL sessions.

**Read more about**[**GlobalSign Root Certificate compatibility**](https://www.globalsign.com/en/ssl-information-center/certificate-authority-root/)**and how it benefits your website security**

The GlobalSign Root Certificate is marked for a number of intended purposes. This makes it a very strong and flexible Root Certificate able to perform all Public Key Infrastructure (PKI) related activities:

         Ensures the identity of a remote computer

         Proves your identity to a remote computer

         Ensures software came from software publisher

         Protects software from alteration after publication

         Protects e-mail messages

         Allows data to be signed with the current time

         Allows data on disk to be encrypted

         Allows secure communication on the Internet

         Permits all key usage policies

         OCSP Signing

GlobalSign provides PKI applications, products and services for all the above security functions. Should your organization have a specific PKI rollout or project, do not hesitate to contact us.

There are many differences between SOAP and REST web services. The important 10 differences between SOAP and REST are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 5) | **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| 6) | SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |

**One-to-Many association on a join table**

Here, a **category** can contain from one to many occurrences of **article**. The **CategoryArticle** is the join table between them. Let’s see how to model this association in Hibernate framework.

The sample application in this tutorial was developed and tested in the following environment:



o    [Hibernate 4.2.2.Final](http://sourceforge.net/projects/hibernate/files/hibernate4/)

o    [JDK 7](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)

o    [Eclipse IDE 4.2 (Juno)](http://www.eclipse.org/downloads/)

o    [MySQL Community Server 5.5.23](http://dev.mysql.com/downloads/mysql/)

o    [MySQL Connector Java driver 5.1.25](http://dev.mysql.com/downloads/connector/j/)

**Table of content:**

1.

1.    [Creating Database Tables](http://www.codejava.net/frameworks/hibernate/hibernate-one-to-many-using-join-table-xml-mapping-example#CreateDatabase)

2.    [Setting up an Eclipse Project](http://www.codejava.net/frameworks/hibernate/hibernate-one-to-many-using-join-table-xml-mapping-example#SetupProject)

3.    [Coding Hibernate Model Classes](http://www.codejava.net/frameworks/hibernate/hibernate-one-to-many-using-join-table-xml-mapping-example#ModelClass)

4.    [Writing Hibernate Mapping Files](http://www.codejava.net/frameworks/hibernate/hibernate-one-to-many-using-join-table-xml-mapping-example#MappingFiles)

5.    [Writing Hibernate Configuration File](http://www.codejava.net/frameworks/hibernate/hibernate-one-to-many-using-join-table-xml-mapping-example#HibernateConfiguration)

6.    [Coding a Test Program](http://www.codejava.net/frameworks/hibernate/hibernate-one-to-many-using-join-table-xml-mapping-example#TestProgram)

# 1. Creating Database Tables

Create a MySQL database called **newsdb** with three tables: **category**, **article**and**CategoryArticle**(join table), by executing the following script in *MySQL Workbench*’s SQL Editor:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | CREATE DATABASE `newsdb`;    use newsdb;    CREATE TABLE `category` (    `category\_id` int(11) NOT NULL AUTO\_INCREMENT,    `name` varchar(45) NOT NULL,    PRIMARY KEY (`category\_id`)  );      CREATE TABLE `article` (    `article\_id` int(11) NOT NULL AUTO\_INCREMENT,    `title` varchar(70) NOT NULL,    `description` varchar(250) NOT NULL,    `keywords` varchar(150) NOT NULL,    `content` text NOT NULL,    PRIMARY KEY (`article\_id`)  );      CREATE TABLE `categoryarticle` (    `category\_id` int(11) NOT NULL,    `article\_id` int(11) NOT NULL,    PRIMARY KEY (`category\_id`,`article\_id`),    KEY `fk\_category` (`category\_id`),    KEY `fk\_article` (`article\_id`),    CONSTRAINT `fk\_article` FOREIGN KEY (`article\_id`) REFERENCES `article` (`article\_id`),    CONSTRAINT `fk\_category` FOREIGN KEY (`category\_id`) REFERENCES `category` (`category\_id`)  ); |

If you are using *MySQL Command Line Client*, type the following command:

**source Path\To\The\Script\File\MySQLscript.sql**

Create the MySQLscript.sql file from the above script or find it in the attached project. The database **newsdb** will have the following structure:

# 2. Setting up an Eclipse Project

Create a Java project in Eclipse IDE called **HibernateOne2ManyJoinTableXMLExample** with the following structure:

We will create/add to this project the following files:



o    Model classes: Category.java and Article.java

o    Hibernate XML mapping files: Category.hbm.xml and Article.hbm.xml

o    Hibernate XML configuration file: hibernate.cfg.xml

o    Test program: ArticlesManager.java

o    Hibernate required JAR libraries and MySQL Connector Java driver:

      I.

         o

        hibernate-core-4.2.2.Final.jar

        hibernate-commons-annotations-4.0.2.Final.jar

        mysql-connector-java-5.1.25-bin.jar

        jboss-transaction-api\_1.1\_spec-1.0.1.Final.jar

        hibernate-jpa-2.0-api-1.0.1.Final.jar

        jboss-logging-3.1.0.GA.jar

        antlr-2.7.7.jar

        dom4j-1.6.1.jar

        javassist-3.15.0-GA.jar



o    MySQL script file: MySQLscript.sql

Find the Hibernate and MySQL jar files from their distribution archives.

If you are new to Hibernate and want to learn more, read this book: [**Hibernate Made Easy: Simplified Data Persistence with Hibernate and JPA (Java Persistence API) Annotations**](http://www.amazon.com/gp/product/0615201954/ref=as_li_qf_sp_asin_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0615201954&linkCode=as2&tag=code0ac-20)

# 3. Coding Hibernate Model Classes

Create two Java files Category.java and Article.java corresponding to the tables **category** and **article**:

File **net\codejava\hibernate\Category.java**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package net.codejava.hibernate;    import java.util.Set;    public class Category {        private long id;      private String name;        private Set<Article> articles;        public Category() {      }        public Category(String name) {  [this.name](http://this.name/) = name;      }        // getters and setters...  } |

File **net\codejava\hibernate\Article.java**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package net.codejava.hibernate;    public class Article {      private long id;      private String title;      private String description;      private String keywords;      private String content;        private Category category;        public Article() {      }        public Article(String title, String description, String keywords,              String content, Category category) {          this.title = title;          this.description = description;          this.keywords = keywords;          this.content = content;          this.category = category;      }        // getters and setters...  } |

**NOTES:**The Category class has a set of articles and the Article class has a reference to its category. This is a typical configuration for a bidirectional one-to-many association. And note that we don’t have to write model class for the join table.

# 4. Writing Hibernate Mapping Files

Create two Hibernate mapping files Category.hbm.xml and Article.hbm.xml with the following content:

File **net\codejava\hibernate\Category.hbm.xml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE hibernate-mapping PUBLIC          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"          "<http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd>">  <hibernate-mapping package="net.codejava.hibernate">      <class name="Category" table="CATEGORY">          <id name="id" column="CATEGORY\_ID">              <generator class="native"/>          </id>          <property name="name" column="NAME" />            <set name="articles" table="CategoryArticle" cascade="all">              <key column="CATEGORY\_ID" not-null="true" />              <many-to-many column="ARTICLE\_ID" class="Article" unique="true"/>          </set>      </class>  </hibernate-mapping> |

File **net\codejava\hibernate\Article.hbm.xml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE hibernate-mapping PUBLIC          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"          "<http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd>">  <hibernate-mapping package="net.codejava.hibernate">      <class name="Article" table="ARTICLE">          <id name="id" column="ARTICLE\_ID">              <generator class="native"/>          </id>          <property name="title" column="TITLE" />          <property name="description" column="DESCRIPTION" />          <property name="keywords" column="KEYWORDS" />          <property name="content" column="CONTENT" />            <join table="CategoryArticle" inverse="true">              <key column="ARTICLE\_ID"/>              <many-to-one name="category" column="CATEGORY\_ID" not-null="true"/>          </join>      </class>  </hibernate-mapping> |

**NOTES:**



o    We specify the join table **CategoryArticle** for both sides, in the **<set>** element of the Category.hbm.xml file and in the **<join>** element of the Article.hbm.xml file.

o    Category.hbm.xml: The attribute **cascade=”all”** tells Hibernate to update changes to the children (**article**) automatically when the parent (**category**) has changed.

o    Article.hbm.xml: The attribute **inverse=”true”** tells Hibernate that the relationship owner is on the reverse side (the join table). Specifying this attribute is required for a one-to-many association.

# 5. Writing Hibernate Configuration File

Write code for the Hibernate configuration file (hibernate.cfg.xml) as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?xml version='1.0' encoding='utf-8'?>  <!DOCTYPE hibernate-configuration PUBLIC          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"          "<http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd>">  <hibernate-configuration>    <session-factory>      <!-- Database connection settings -->      <property name="connection.driver\_class">com.mysql.jdbc.Driver</property>      <property name="connection.url">jdbc:mysql://localhost:3306/newsdb</property>      <property name="connection.username">root</property>      <property name="connection.password">secret</property>      <property name="dialect" |

Wall mart 🡺

1.       IOC Pattern development

2.       SQL Query for LAZY Loading of  of One to many relationship  : single sql query with joins

3.       Thread Priority and Method 2-3 at same time calling on diff or same object

4.       Calling same method on diff objects

5.       Hibernate Template and JMS Template

6.       Why string is Immutable

7.       Cohesive ness in JAVA

8.       Which is better between arraylist and linked list

Suppose you are given a Linked List as 45-> 123-> 87-> 11-> 53-> 24-> 412-> 22. And we have to find the nth node from the end.

Input:- n = 3

Output:- 24

BRUTE FORCE APPROACH:-

In this method, start with the first node and count how many nodes are there after that node. If the number of nodes are < n-1, then return saying, “fewer number of nodes in the list.” If the number of nodes are > n-1 then go to the next node. Continue this until the number of nodes after the current node are n-1.

Time Complexity:- O(n2), for scanning the remaining list (from the current node) for each node.

Space Complexity:- O(1)

IMPROVING THE COMPLEXITY:-

The complexity of the above problem can be improved by using a hash table. Consider the list:- 5-> 1-> 17-> 4-> NULL.

In this approach, create a hash table whose entries are <position of node, node address>. That means, key is the position of the node in the list and the value is the address of that node.

Position in List

Address of node

1

Address of 5 node

2

Address of 1 node

3

Address of 17 node

4

Address of 4 node

By the time we traverse the complete list(for creating hash table), we an find the list length. Let us say, the list length is M. To find the nth term from the end of the Linked List, we can convert this to M-n+1thfrom beginning. Since we already know the length of the list, its just a matter of returning M-n+1th key value from the hash table.

Time Complexity:- Time for creating hash table. Therefore T(M) = O(M).

Space Complexity:- O(M), because we are creating a hash table of size M.

AN APPROACH WITHOUT USING THE HASH TABLE

In the above solution, what actually we are doing is finding the size of the Linked List. That means, we are using hash table to find the size of the Linked list. We can find the length of the linked list just after starting at the head node and traversing the list. So, we can find the length of the list without creating the hash table. After finding the length compute M-n+1 and with one more scan we can get M-n+1thterm from the beginning.

Time Complexity:- Time for finding length + Time for finding M-n+1th node from beginning, Therefore T(n) = O(n) + O(n) ≈ O(n).

Space Complexity:- O(1), since no hash table is created.

SOLVING THE PROBLEM IN ONE SCAN (MOST EFFICIENT)

Use 2 pointers:-

§ pNthNode

§ pTemp

Both the nodes point to the HEAD of the list. pNthNode starts moving only after pTemp has moved n nodes forward. From there, both nodes move forward until pTemp reaches NULL.

pNthNode now points at the nth node from the end.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

/\*

Defining a function to find the nth node from the end.

\*/

struct node \* nthNodeFromEnd(struct node \* head, int n)

{

// we need to find the nth node from the end.

struct node \* pTemp = head;

struct node \* pNthNode = head;

int nCurrentElement = 0;

int counter = 0;

while(counter != n)

{

pTemp = pTemp -> next;

counter++;

}

while(pTemp != NULL)

{

pNthNode = pNthNode -> next;

pTemp = pTemp -> next;

}

return pNthNode;

}

Time Complexity:- O(n)

Space Complexity:- O(1)

Spring Bean Configuration File for Hibernate 3 Integration

Let’s first look at the spring bean configurations we need for Hibernate 3 integration, we will look into detail later on.

|  |  |
| --- | --- |
| spring.xml | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | <?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:aop="<http://www.springframework.org/schema/aop>"      xmlns:tx="<http://www.springframework.org/schema/tx>"      xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsd>  <http://www.springframework.org/schema/aop> <http://www.springframework.org/schema/aop/spring-aop-4.0.xsd>  <http://www.springframework.org/schema/tx> <http://www.springframework.org/schema/tx/spring-tx-4.0.xsd>">        <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"          destroy-method="close">          <property name="driverClassName" value="com.mysql.jdbc.Driver" />          <property name="url" value="jdbc:mysql://localhost:3306/TestDB" />          <property name="username" value="pankaj" />          <property name="password" value="pankaj123" />      </bean>    <!-- Hibernate 3 XML SessionFactory Bean definition-->  <!--     <bean id="hibernate3SessionFactory"          class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">          <property name="dataSource" ref="dataSource" />          <property name="mappingResources">              <list>                  <value>person.hbm.xml</value>              </list>          </property>          <property name="hibernateProperties">              <value>                  hibernate.dialect=org.hibernate.dialect.MySQLDialect              </value>          </property>      </bean> -->    <!-- Hibernate 3 Annotation SessionFactory Bean definition-->      <bean id="hibernate3AnnotatedSessionFactory"          class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">          <property name="dataSource" ref="dataSource" />          <property name="annotatedClasses">              <list>                  <value>com.journaldev.model.Person</value>              </list>          </property>          <property name="hibernateProperties">              <props>                  <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>                  <prop key="hibernate.current\_session\_context\_class">thread</prop>                  <prop key="hibernate.show\_sql">false</prop>              </props>          </property>      </bean>        <bean id="personDAO" class="com.journaldev.dao.PersonDAOImpl">          <property name="sessionFactory" ref="hibernate3AnnotatedSessionFactory" />      </bean>  </beans> |

There are two ways we can provide database connection details to Hibernate, first by passing everything inhibernateProperties and second by creating a DataSource and then passing it to hibernate. I prefer the second approach, that’s why we have Apache Commons DBCP dependency to create a BasicDataSource by setting database connection properties.

For Spring and Hibernate 3 integration, **Spring ORM** provides two classes –org.springframework.orm.hibernate3.LocalSessionFactoryBean when hibernate mappings are XML based andorg.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean for annotations based mapping. I have provided simple bean configuration of LocalSessionFactoryBean in comments, if you are using XML based mappings. AnnotationSessionFactoryBean extends LocalSessionFactoryBean class, so it has all the basic properties for hibernate integration.

The properties are self understood and mostly hibernate related, so I will not go into much detail for them. But if you are wondering from where **hibernateProperties**, **annotatedClasses** are coming, you need to look into the bean class source code.

<?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:aop="<http://www.springframework.org/schema/aop>"

    xmlns:tx="<http://www.springframework.org/schema/tx>"

    xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://www.springframework.org/schema/aop> <http://www.springframework.org/schema/aop/spring-aop-4.0.xsd>

<http://www.springframework.org/schema/tx> <http://www.springframework.org/schema/tx/spring-tx-4.0.xsd>">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"

        destroy-method="close">

        <property name="driverClassName" value="com.mysql.jdbc.Driver" />

        <property name="url" value="jdbc:mysql://localhost:3306/TestDB" />

        <property name="username" value="pankaj" />

        <property name="password" value="pankaj123" />

    </bean>

<!-- Hibernate 4 SessionFactory Bean definition -->

<bean id="hibernate4AnnotatedSessionFactory"

        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

        <property name="dataSource" ref="dataSource" />

        <property name="annotatedClasses">

            <list>

                <value>com.journaldev.model.Person</value>

            </list>

        </property>

        <property name="hibernateProperties">

            <props>

                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

                <prop key="hibernate.current\_session\_context\_class">thread</prop>

                <prop key="hibernate.show\_sql">false</prop>

            </props>

        </property>

    </bean>

    <bean id="personDAO" class="com.journaldev.dao.PersonDAOImpl">

        <property name="sessionFactory" ref="hibernate4AnnotatedSessionFactory" />

    </bean>

</beans>

### **REST and SOAP Web Service Interview Questions**

Posted by [Dinesh Rajput](https://plus.google.com/117630637804698317816)

In this interview questions tutorial we will explain most asking interviews questions on the web services like SOAP, REST etc and its protocol support. REST is getting popular day by day and replacing SOAP web services which was standard earlier and Interviewer expect you to know about REST and how it work.  
  
**Define Web Service?**  
A web service is a kind of software that is accessible on the Internet. It makes use of the XML messaging system and offers an easy to understand, interface for the end users.  
  
**What is REST and RESTful web services ?**  
REST stands for **REpresentational State Transfer (REST)** its a relatively new concept of writing web services which enforces a stateless client server design where web services are treated as resource and can be accessed and identified by there URL unlike SOAP web services which were defined by WSDL.  
  
Web services written by apply REST Architectural concept are called RESTful web services which focus on System resources and how state of Resource should be transferred over http protocol to a different clients written in different languages. In RESTful web services http methods like GET, PUT, POST and DELETE can can be used to perform CRUD operations.  
  
**What is differences between RESTful web services and SOAP web services ?**  
Though both RESTful web series and SOAP web service can operate cross platform they are architecturally different to each other, here is some of differences between REST and SOAP:  
  
1) REST is more simple and easy to use than SOAP  
2) REST uses HTTP protocol for producing or consuming web services while SOAP uses XML.  
3) REST is lightweight as compared to SOAP and preferred choice in mobile devices and PDA's.  
4) REST supports different format like text, JSON and XML while SOAP only support XML.  
5) REST web services call can be cached to improve performance.  
  
**What is Restlet framework ?**  
Restlet is leading RESTful web framework for Java applications is used to build RESTFul web services it has two part Restlet API and a Restlet implementation much like Servlet specification. There are many implementation of Restlet framework available you just need to add there jar in your classpath to use them. By using Restlet web framework you can write client and server.  
  
**What is Resource in REST framework ?**  
it represent a "resource" in REST architecture. on RESTLET API it has life cycle methods like init(), handle() and release() and contains a Context, Request and Response corresponding to specific target resource. This is now deprecated over ServerResource class and you should use that. see Restlet documentation for more details.  
  
**Can you use Restlet without any web-container ?**  
Yes, Restlet framework provide default server which can be used to handle service request in web container is not available.  
  
**What are the tools used for creating RESTFull web services ?**  
You can use AJAX(Asynchronous JavaScript with XAML) and Direct Web Removing to consume web serives in web application. Both Eclipse and NetBeans also supported development of RESTFul services.  
  
**How to display custom error pages using RestFull web services ?**  
In order to customize error you need to extend StatusService and implement getRepresentation(Status, Request, Response) method with your custom code now assign instance of your CustomStatusService to appropriate "statusService property".  
  
**Which HTTP methods are supported by RestFull web services ?**  
Another common REST interview questioning RESTFul web service each Resource supports GET, POST, PUT and DELETE http methods.GET is mapped to represent(), POST - acceptRepresentation(), PUT- storeRepresentation and DELET for rmeoveRepresentation.  
  
**What is difference between top-down and bottom-up approach of developing web services ?**  
In top-down approach first WSDL document is created and than Java classes are developed based on WSDL contract, so if WSDL contract changes you got to change your Java classes while in case of bottom up approach of web service development you first create Java code and then use annotations like @WebService to specify contract or interface and WSDL field will be automatically generated from your build.  
  
**Define SOAP?**  
SOAP is an XML based protocol to transfer between computers.  
  
**Define WSDL?**  
It means Web Services Description Language. It is basically the service description layer in the web service protocol stock. The Service Description layer describes the user interface to a web service.  
  
**Differentiate between a SOA and a Web service?**  
SOA is a design and architecture to implement other services. SOA can be easily implemented using various protocols such as HTTP, HTTPS, JMS, SMTP, RMI, IIOP, RPC etc. While Web service, itself is an implemented technology. In fact one can implement SOA using the web service.  
  
**Discuss various approaches to develop SOAP based web service?**  
We can develop SOAP based web service with two different types of approaches such as contract-first and contract-last. In the first approach, the contract is defined first and then the classes are derived from the contract while in the later one, the classes are defined first and then the contract is derived from these classes.  
  
**If you have to choose one approach, then what will be your choice?**  
In my point of view, the first approach that is the contract-first approach is more feasible as compared to the second one but still it depends on other factors too.  
  
**What are the types of information included in SOAP header?**  
Header of SOAP contains information like that,  
1. In SOAP header client should handle authentication and transaction.  
2. The SOAP message should process by client.  
3. EncodingStyle is also has in header.  
  
**What are the disadvantages of SOAP?**  
Some disadvantages .  
1. It is much slower than middleware technologies.  
2. Because we used HTTP for transporting messages and not use to defined ESB or WS-Addressing interaction of parties over a message is fixed.  
3. Application protocol level is problematic because usability of HTTP for different purposes is not present.

**Erasure of Generic Types**

During the type erasure process, the Java compiler erases all type parameters and replaces each with its first bound if the type parameter is bounded, or Object if the type parameter is unbounded.

Consider the following generic class that represents a node in a singly linked list:

public class Node<T> {

    private T data;

    private Node<T> next;

    public Node(T data, Node<T> next) }

        this.data = data;

        this.next = next;

    }

   public T getData() { return data; }

    // ...

}

Because the type parameter T is unbounded, the Java compiler replaces it with Object:

public class Node {

    private Object data;

    private Node next;

    public Node(Object data, Node next) {

        this.data = data;

        this.next = next;

    }

    public Object getData() { return data; }

    // ...

}

In the following example, the generic Node class uses a bounded type parameter:

public class Node<T extends Comparable<T>> {

    private T data;

    private Node<T> next;

    public Node(T data, Node<T> next) {

        this.data = data;

        this.next = next;

    }

    public T getData() { return data; }

    // ...

}

The Java compiler replaces the bounded type parameter T with the first bound class, Comparable:

public class Node {

    private Comparable data;

    private Node next;

    public Node(Comparable data, Node next) {

        this.data = data;

        this.next = next;

    }

    public Comparable getData() { return data; }

    // ...

}

Difference between ConcurrentHashMap, Hashtable and Synchronized Map in Java

Though all three collection classes are thread-safe and can be used in multi-threaded, concurrent Java application, there is significant difference between them, which arise from the fact that how they achieve their thread-safety. Hashtable is a legacy class from JDK 1.1 itself, which uses synchronized methods to achieve thread-safety. All methods of Hashtable are synchronized which makes them quite slow due to contention if number of thread increases. Synchronized Map is also not very different than Hashtable and provides similar performance in concurrent Java programs. Only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create it's synchronized version by using Collections.synchronizedMap() method. On the other hand, ConcurrentHashMap is especially designed for concurrent use i.e. more than one thread. By default it simultaneously allows 16 threads to read and write from Map without any external synchronization. It is also very scalable because of stripped locking technique used in [internal implementation of ConcurrentHashMap](http://javarevisited.blogspot.sg/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) class. Unlike Hashtable and Synchronized Map, it never locks whole Map, instead it divides the map in segments and locking is done on those. Though it perform better if number of reader threads is greater than number of writer threads.

Read more: <http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html#ixzz3lUixbLH3>  
  
Read more: <http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html#ixzz3lUinnQgy>

n Spring framework, you can wire beans automatically with auto-wiring feature. To enable it, just define the “**autowire**” attribute in <bean>.

         <bean id="customer" class="com.mkyong.common.Customer" autowire="byName" />

In Spring, 5 Auto-wiring modes are supported.

* no – Default, no auto wiring, set it manually via “ref” attribute
* byName – Auto wiring by property name. If the name of a bean is same as the name of other bean property, auto wire it.
* byType – Auto wiring by property data type. If data type of a bean is compatible with the data type of other bean property, auto wire it.
* constructor – byType mode in constructor argument.
* autodetect – If a default constructor is found, use “autowired by constructor”; Otherwise, use “autowire by type”.

## **Examples**

A Customer and Person object for auto wiring demonstration.

package com.mkyong.common;

public class Customer

{

         private Person person;

         public Customer(Person person) {

                  this.person = person;

         }

         public void setPerson(Person person) {

                  this.person = person;

         }

         //...

}

package com.mkyong.common;

public class Person

{

         //...

}

## **1. Auto-Wiring ‘no’**

This is the default mode, you need to wire your bean via ‘ref’ attribute.

         <bean id="customer" class="com.mkyong.common.Customer">

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

         </bean>

         <bean id="person" class="com.mkyong.common.Person" />

## **2. Auto-Wiring ‘byName’**

Auto-wire a bean by property name. In this case, since 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="com.mkyong.common.Customer" autowire="byName" />

         <bean id="person" class="com.mkyong.common.Person" />

See full example – [Spring Autowiring by Name](http://www.mkyong.com/spring/spring-autowiring-by-name/).

## **3. Auto-Wiring ‘byType’**

Auto-wire a bean by property data type. In this case, since 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="com.mkyong.common.Customer" autowire="byType" />

         <bean id="person" class="com.mkyong.common.Person" />

See full example – [Spring Autowiring by Type](http://www.mkyong.com/spring/spring-autowiring-by-type/).

## **4. Auto-Wiring ‘constructor’**

Auto-wire a bean by property data type in constructor argument. In this case, since 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="com.mkyong.common.Customer" autowire="constructor" />

         <bean id="person" class="com.mkyong.common.Person" />

See full example – [Spring Autowiring by Constructor](http://www.mkyong.com/spring/spring-autowiring-by-constructor/).

## **5. Auto-Wiring ‘autodetect’**

If a default constructor is found, uses “constructor”; Otherwise, uses “byType”. In this case, since there is a default constructor in “Customer” class, so, Spring auto wired it via constructor method – “public Customer(Person person)“.

         <bean id="customer" class="com.mkyong.common.Customer" autowire="autodetect" />

         <bean id="person" class="com.mkyong.common.Person" />

See full example – [Spring Autowiring by AutoDetect](http://www.mkyong.com/spring/spring-autowiring-by-autodetect/).

**Note**  
It’s always good to combine both ‘auto-wire’ and ‘dependency-check’ together, to make sure the property is always auto-wire successfully.

         <bean id="customer" class="com.mkyong.common.Customer"

                          autowire="autodetect" dependency-check="objects />

         <bean id="person" class="com.mkyong.common.Person" />

## **Conclusion**

Here is the updated program that will always output count value as 8 because **AtomicInteger***incrementAndGet()* atomically increments the current value by one.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | package com.journaldev.concurrency;    import java.util.concurrent.atomic.AtomicInteger;      public class JavaAtomic {        public static void main(String[] args) throws InterruptedException {            ProcessingThread pt = new ProcessingThread();          Thread t1 = new Thread(pt, "t1");          t1.start();          Thread t2 = new Thread(pt, "t2");          t2.start();          t1.join();          t2.join();          System.out.println("Processing count=" + pt.getCount());      }    }      class ProcessingThread implements Runnable {      private AtomicInteger count = new AtomicInteger();          @Override      public void run() {          for (int i = 1; i < 5; i++) {              processSomething(i);              count.incrementAndGet();          }      }          public int getCount() {          return this.count.get();      }          private void processSomething(int i) {          // processing some job          try {              Thread.sleep(i \* 1000);          } catch (InterruptedException e) {              e.printStackTrace();          }      }    } |

Benefits of using Atomic Concurrency classes is that we don’t need to worry about synchronization at each and every place we are dealing with integers and it’s assumed to be more efficient that synchronization which involves locking resources.

# serialVersionUID in Java Serialization

serialVersionUID is used to ensure that during deserialization the same class (that was used during serialize process) is loaded. This is a one line definition to explain why a serialVersionUID is used?

Apart from the above definition there are quite  a few things to learn from this serialVersionUID. As per javadocs, following is format of serialVersionUID:

## **serialVersionUID Syntax**

ANY-ACCESS-MODIFIER static final long serialVersionUID = 42L;

* serialVersionUID is a static final field. You can assign any number of your choice to it. Later I will explain the significance of these two statements.

## **Why serialVersionUID?**

Lets start with annoying warning message you get in your IDE when you declare a class as Serializable.

*The serializable class Lion does not declare a static final serialVersionUID field of type long*

Most of us used to ignore this message as we always do for a warning. My general note is, always pay attention to the java warning messages. It will help you to learn a lot of fundamentals.

serialVersionUID is a must in serialization process. But it is optional for the developer to add it in java source file. If you are not going to add it in java source file, serialization runtime will generate a serialVersionUID and associate it with the class. The serialized object will contain this serialVersionUID along with other data.

Even though serialVersionUID is a static field, it gets serialized along with the object. This is one exception to the general serialization rule that, “static fields are not serialized”.

## **How serialVersionUID is generated?**

serialVersionUID is a 64-bit hash of the class name, interface class names, methods and fields. Serialization runtime generates a serialVersionUID if you do not add one in source. Refer this [link for the algorithm to generate serialVersionUID](http://docs.oracle.com/javase/6/docs/platform/serialization/spec/class.html#4100).

It is advised to have serialVersionUID as unique as possible. Thats why the java runtime chose to have such a complex algorithm to generate it.

If you want help in generating it, jdk tools provides a tool named serialver. Use**serialver -show** to start the gui version of the tool as shown below.

## **How serialVersionUID works?**

When an object is serialized, the serialVersionUID is serialized along with the other contents.

Later when that is deserialized, the serialVersionUID from the deserialized object is extracted and compared with the serialVersionUID of the loaded class.

If the numbers do not match then, InvalidClassException is thrown.

If the loaded class is not having a serialVersionUID declared, then it is automatically generated using the same algorithm as before.

## **Strongly recommended to declare serialVersionUID**

Javadocs says,

*“the default serialVersionUID computation is highly sensitive to class details that may vary depending on compiler implementations, and can thus result in unexpected InvalidClassExceptions during deserialization”*

Now you know why we should declare a serialVersionUID.

Not only declaring a serialVersionUID is sufficient. You must do the following two things carefully. Otherwise it defeats the purpose of having the serialVersionUID.

serialVersionUID should be maintained. As and when you change anything in the class, you should upgrade the serailVersionUID.  
Try your best to declare a unique serialVersionUID.

## **Demonstrate serialVersionUID**

Initial class to be serialized has a serialVersionUID as 1L.

import java.io.Serializable;

public class Lion implements Serializable {

  private static final long serialVersionUID = 1L;

  private String sound;

  public Lion(String sound) {

    this.sound = sound;

  }

  public String getSound() {

    return sound;

  }

}

Test serialVersionUID:

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

public class SerialVersionUIDTest {

  public static void main(String args[]) throws IOException, ClassNotFoundException {

    Lion leo = new Lion("roar");

    // serialize

    System.out.println("Serialization done.");

    FileOutputStream fos = new FileOutputStream("serial.out");

    ObjectOutputStream oos = new ObjectOutputStream(fos);

    oos.writeObject(leo);

    // deserialize

    FileInputStream fis = new FileInputStream("serial.out");

    ObjectInputStream ois = new ObjectInputStream(fis);

    Lion deserializedObj = (Lion) ois.readObject();

    System.out.println("DeSerialization done. Lion: " + deserializedObj.getSound());

  }

}

### Output:

Serialization done.

DeSerialization done. Lion: roar

Now change serialVersionUID to 2L in Lion class.

  private static final long serialVersionUID = 2L;

Comment the “serialize” block (4 lines of code) in SerialVersionUIDTest. Now run it and you will get the following exception.

1. Serialized Lion with serialVersionUID with 1L.
2. Changed serialVersionUID to 2L and compiled and loaded the class.
3. Deserialize the already serialized object and load it with the latest class.
4. We get exception as serialVersionUID is not matching.

Exception in thread "main" java.io.InvalidClassException: Lion; local class incompatible: **stream classdesc serialVersionUID = 1, local class serialVersionUID = 2**

        at java.io.ObjectStreamClass.initNonProxy(Unknown Source)

        at java.io.ObjectInputStream.readNonProxyDesc(Unknown Source)

        at java.io.ObjectInputStream.readClassDesc(Unknown Source)

        at java.io.ObjectInputStream.readOrdinaryObject(Unknown Source)

        at java.io.ObjectInputStream.readObject0(Unknown Source)

        at java.io.ObjectInputStream.readObject(Unknown Source)

        at SerialVersionUIDTest.main(SerialVersionUIDTest.java

**public class**Base {  
**private int**a;  
**private int**b;  
  
**public**Base(**int**a, **int**b) {  
**this**.a = a;  
**this**.b = b;  
}  
  
/\*  
\* Add Getters and Setters...  
\*/  
}  
  
This has a derived class, Derived4  
  
**public class**Derived4 **extends**Base **implements**Serializable {  
**private int**c;  
  
**public**Derived4(**int**a, **int**b, **int**c) {  
**super**(a,b);  
**this**.c = c;  
}  
  
**public**Derived4(DataToken d) {  
**super**(d.getA(),d.getB());  
c = d.getC();  
}  
  
/\*  
\* Add Getters and Setters...  
\*/  
  
**private**Object writeReplace() **throws**ObjectStreamException {  
Derived4DataToken d = **new**Derived4DataToken();  
d.setA(**this**.getA());  
d.setB(**this**.getB());  
d.setC(**this**.getC());  
**return**d;  
}  
}  
  
Note the writeReplace()? Now, lets see the DTO class: Derived4DataToken  
  
**public class**Derived4DataToken **implements**Serializable {  
**private int**a;  
**private int**b;  
**private int**c;  
  
**private static final long**serialVersionUID = -2835599374077375317L;  
  
/\*  
\* Add Getters and Setters...  
\*/  
  
**private**Object readResolve() **throws**ObjectStreamException {  
**return new**Derived4(**this**);  
}  
}  
  
So now, we can go ahead and read and write the class, exactly as normal. Note that if Derived4DataToken had not implemented readResolve(), reading the data stream would have resulted in a Derived4DataToken instance, not a Derived4 instance.  
  
ByteArrayOutputStream os = **new**ByteArrayOutputStream();  
Derived4 d1 = **new**Derived4(10, 20, 30);  
ObjectOutputStream oos = **new**ObjectOutputStream(os);  
oos.writeObject(d1);  
oos.flush();  
Derived4 d2;  
ObjectInputStream ois = **new**ObjectInputStream(**new**ByteArrayInputStream(os.toByteArray()));  
d2 = (Derived4)ois.readObject();  
out.println("Derived4=> a : b : c :: " + d2.getA() + " : " + d2.getB() + " : " + d2.getC());  
  
Finally, let me add that even if you are not making your class Serializable, if you want to be safe such that child classes can be Serializable then the cleanest solution is to have a child visible empty constructor, it will save you much heartache later.

# Hibernate Many-To-One Mapping Tutorial

from [{{ articles[0].portal.name }} Zone](https://dzone.com/tutorials/java/hibernate/hibernate-example/%7B%7B%20articles%5b0%5d.portal.url%20%7D%7D) - check it out...

[Meyyappan Muthuraman](https://dzone.com/users/1031353/eswar_vaannila.html)

{{ articles[0].views | formatCount}} Views

Send

Tweet

*{{articles[0].nComments}}*

In this example you will learn how to map many-to-one relationship using Hibernate. Consider the following relationship between Student and Address entity.

According to the relationship many students can have the same address.

To create this relationship you need to have a STUDENT and ADDRESS table. The relational model is shown below.

To create the STUDENT and ADDRESS table you need to create the following hibernate mapping files.

Student.hbm.xml is used to create the STUDENT table.

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">

<hibernate-mapping>

    <class name="com.vaannila.student.Student" table="STUDENT">

        <meta attribute="class-description">This class contains student details.</meta>

        <id name="studentId" type="long" column="STUDENT\_ID">

            <generator class="native" />

        </id>

        <property name="studentName" type="string" length="100" not-null="true" column="STUDENT\_NAME" />

        <many-to-one name="studentAddress" class="com.vaannila.student.Address" column="STUDENT\_ADDRESS" cascade="all" not-null="true" />

    </class>

</hibernate-mapping>

The many-to-one element is used to create the many-to-one relationship between the Student and Address entities. The cascade option is used to cascade the required operations to the associated entity. If the cascade option is set to all then all the operations will be cascaded. For instance when you save a Student object, the associated Address object will also be saved automatically.

Address.hbm.xml is used to create the ADDRESS table.

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">

<hibernate-mapping>

        <class name="com.vaannila.student.Address" table="ADDRESS">

               <meta attribute="class-description">This class contains the student's address

                       details.</meta>

               <id name="addressId" type="long" column="ADDRESS\_ID">

                       <generator class="native" />

               </id>

               <property name="street" column="ADDRESS\_STREET" type="string" length="250" />

               <property name="city" column="ADDRESS\_CITY" type="string" length="50" />

               <property name="state" column="ADDRESS\_STATE" type="string" length="50" />

               <property name="zipcode" column="ADDRESS\_ZIPCODE" type="string" length="10" />

        </class>

</hibernate-mapping>

Now create the hibernate configuration file and add all the mapping files.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

               "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

               "<http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd>">

<hibernate-configuration>

    <session-factory>

        <property name="hibernate.connection.driver\_class"> org.hsqldb.jdbcDriver </property>

        <property name="hibernate.connection.url"> jdbc:hsqldb:hsql://localhost</property>

        <property name="hibernate.connection.username">sa</property>

        <property name="connection.password"></property>

        <property name="connection.pool\_size">1</property>

        <property name="hibernate.dialect"> org.hibernate.dialect.HSQLDialect </property>

        <property name="show\_sql">true</property>

        <property name="hbm2ddl.auto">create-drop</property>

        <mapping resource="com/vaannila/student/Student.hbm.xml"/>

        <mapping resource="com/vaannila/student/Address.hbm.xml"/>

    </session-factory>

</hibernate-configuration>

After creating the configuration file, generate java class files using Hibernate Tools.(To generate code using Hibernate Tools refer this [example](http://vaannila.com/hibernate/hibernate-example/hibernate-tools-1.html) )

The following classes will be generated.

package com.vaannila.student;

// Generated Sep 3, 2009 7:20:37 PM by Hibernate Tools [3.2.4.GA](http://3.2.4.ga/)

/\*\*

\* This class contains student details.

\*/

public class Student implements java.io.Serializable {

        private long studentId;

        private String studentName;

        private Address studentAddress;

        public Student() {

        }

        public Student(String studentName, Address studentAddress) {

               this.studentName = studentName;

               this.studentAddress = studentAddress;

        }

        public long getStudentId() {

               return this.studentId;

        }

        public void setStudentId(long studentId) {

               this.studentId = studentId;

        }

        public String getStudentName() {

               return this.studentName;

        }

        public void setStudentName(String studentName) {

               this.studentName = studentName;

        }

        public Address getStudentAddress() {

               return this.studentAddress;

        }

        public void setStudentAddress(Address studentAddress) {

               this.studentAddress = studentAddress;

        }

}

package com.vaannila.student;

// Generated Sep 3, 2009 7:20:37 PM by Hibernate Tools [3.2.4.GA](http://3.2.4.ga/)

/\*\*

\* This class contains the student's address

\*                     details.

\*/

public class Address implements java.io.Serializable {

        private long addressId;

        private String street;

        private String city;

        private String state;

        private String zipcode;

        public Address() {

        }

        public Address(String street, String city, String state, String zipcode) {

               this.street = street;

               this.city = city;

               this.state = state;

               this.zipcode = zipcode;

        }

        public long getAddressId() {

               return this.addressId;

        }

        public void setAddressId(long addressId) {

               this.addressId = addressId;

        }

        public String getStreet() {

               return this.street;

        }

        public void setStreet(String street) {

               this.street = street;

        }

        public String getCity() {

               return this.city;

        }

        public void setCity(String city) {

               this.city = city;

        }

        public String getState() {

               return this.state;

        }

        public void setState(String state) {

               this.state = state;

        }

        public String getZipcode() {

               return this.zipcode;

        }

        public void setZipcode(String zipcode) {

               this.zipcode = zipcode;

        }

}

package com.vaannila.student;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.vaannila.util.HibernateUtil;

public class Main {

        public static void main(String[] args) {

               Session session = HibernateUtil.getSessionFactory().openSession();

               Transaction transaction = null;

               try {

                       transaction = session.beginTransaction();

                       Address address = new Address("OMR Road", "Chennai", "TN", "600097");

                       //By using cascade=all option the address need not be saved explicitly when the student object is persisted the address will be automatically saved.

            //session.save(address);

                       Student student1 = new Student("Eswar", address);

                       Student student2 = new Student("Joe", address);

                       session.save(student1);

                       session.save(student2);

                       transaction.commit();

               } catch (HibernateException e) {

                       transaction.rollback();

                       e.printStackTrace();

               } finally {

                       session.close();

               }

        }

}

On executing the Main class you will see the following output.

The Student table has two records.

The Address table has one record.

Both the student records points to the same address record, this illustrates the many-to-one mapping.

The folder structure of the example is shown below.

You can download the source code of this example here.

Source :[Download](https://dzone.com/sites/all/files/HibernateExample5.zip)

### **{{ editionName }}**

**{{ date }}**

* {{ node.blurb }}

## {{ ::node.title }}

{{ parent.articleDate | date:'MMM dd, yyyy' }} {{ parent.linkDate | date:'MMM dd, yyyy' }}

Our singleton class is:

|  |
| --- |
| public class DemoSingleton implements Serializable {      private volatile static DemoSingleton instance = null;        public static DemoSingleton getInstance() {          if (instance == null) {              instance = new DemoSingleton();          }          return instance;      }        private int i = 10;        public int getI() {          return i;      }        public void setI(int i) {          this.i = i;      }  } |

Lets serialize this class and de-serialize it after making some changes:

|  |
| --- |
| public class SerializationTest {      static DemoSingleton instanceOne = DemoSingleton.getInstance();        public static void main(String[] args) {          try {              // Serialize to a file              ObjectOutput out = new ObjectOutputStream(new FileOutputStream(                      "filename.ser"));              out.writeObject(instanceOne);              out.close();                instanceOne.setI(20);                // Serialize to a file              ObjectInput in = new ObjectInputStream(new FileInputStream(                      "filename.ser"));              DemoSingleton instanceTwo = (DemoSingleton) in.readObject();              in.close();                System.out.println(instanceOne.getI());              System.out.println(instanceTwo.getI());            } catch (IOException e) {              e.printStackTrace();          } catch (ClassNotFoundException e) {              e.printStackTrace();          }      }  }    Output:  20  10 |

Unfortunately, both variables have different value of variable “i”. Clearly, there are two instances of our class. So, again we are in same problem of multiple instances in application.  
To solve this issue, we need to include readResolve() method in our DemoSingleton class. This method will be invoked when you will de-serialize the object. Inside this method, you must return the existing instance to ensure single instance application wide.

|  |
| --- |
| public class DemoSingleton implements Serializable {      private volatile static DemoSingleton instance = null;        public static DemoSingleton getInstance() {          if (instance == null) {              instance = new DemoSingleton();          }          return instance;      }        protected Object readResolve() {          return instance;      }        private int i = 10;        public int getI() {          return i;      }        public void setI(int i) {          this.i = i;      }  } |

Now when you execute the class SerializationTest, it will give you correct output.

|  |
| --- |
| 20  20 |

Adding serial version id

So far so good. Till now, we have solved the problem of synchronization and serialization both. Now, we are just one step behind our correct and complete implementation. And missing part is serial version id.

This is required in condition when you class structure can change in between you serialize the instance and go again to de-serialize it. Changed structure of class will cause JVM to give exception while de-serializing process.

|  |
| --- |
| java.io.InvalidClassException: singleton.DemoSingleton; local class incompatible: stream classdesc serialVersionUID = 5026910492258526905, local class serialVersionUID = 3597984220566440782  at java.io.ObjectStreamClass.initNonProxy(Unknown Source)  at java.io.ObjectInputStream.readNonProxyDesc(Unknown Source)  at java.io.ObjectInputStream.readClassDesc(Unknown Source)  at java.io.ObjectInputStream.readOrdinaryObject(Unknown Source)  at java.io.ObjectInputStream.readObject0(Unknown Source)  at java.io.ObjectInputStream.readObject(Unknown Source)  at singleton.SerializationTest.main(SerializationTest.java:24) |

This problem can be solved only by adding a unique serial version id to class. It will prevent the compiler to throw the exception by telling that both classes are same, and will load the available instance variables only.

Conclusion

After having discussed so many possible approaches and other possible error cases, i will recommend you below code template to design your singleton class which shall ensure only one instance of class in whole  application in all above discussed scenarios.

|  |
| --- |
| public class DemoSingleton implements Serializable {      private static final long serialVersionUID = 1L;        private DemoSingleton() {          // private constructor      }        private static class DemoSingletonHolder {          public static final DemoSingleton INSTANCE = new DemoSingleton();      }        public static DemoSingleton getInstance() {          return DemoSingletonHolder.INSTANCE;      }        protected Object readResolve() {          return getInstance();      }  } |

I hope, this post has enough information to make you understand the most common approaches for singleton pattern. Let me know of you thoughts please.

Differences between session vs session factory - Hibernate?

1.    SessionFactory objects are one per application and Session objects are one per client.

2.    SessionFactory is to create and manage Sessions. Session is to provide a CRUD interface for mapped classes, and also access to the more versitile Criteria API.

3.    SessionFactory is thread safe where as Session is not thread safe

public class SingletonSerializedTest {

    public static void main(String[] args) throws FileNotFoundException, IOException, ClassNotFoundException {

        SerializedSingleton instanceOne = SerializedSingleton.getInstance();

        ObjectOutput out = new ObjectOutputStream(new FileOutputStream(

                "filename.ser"));

        out.writeObject(instanceOne);

        out.close();

        //deserailize from file to object

        ObjectInput in = new ObjectInputStream(new FileInputStream(

                "filename.ser"));

        SerializedSingleton instanceTwo = (SerializedSingleton) in.readObject();

        in.close();

        System.out.println("instanceOne hashCode="+instanceOne.hashCode());

        System.out.println("instanceTwo hashCode="+instanceTwo.hashCode());

    }

}

Output of the above program is;

|  |  |
| --- | --- |
| 1  2 | instanceOne hashCode=2011117821  instanceTwo hashCode=109647522 |

So it destroys the singleton pattern, to overcome this scenario all we need to do it provide the implementation of readResolve() method.

|  |  |
| --- | --- |
| 1  2  3 | protected Object readResolve() {      return getInstance();  } |

After this you will notice that hashCode of both the instances are same in test program.

import java.lang.reflect.Constructor;

public class ReflectionSingletonTest {

    public static void main(String[] args) {

        EagerInitializedSingleton instanceOne = EagerInitializedSingleton.getInstance();

        EagerInitializedSingleton instanceTwo = null;

        try {

            Constructor[] constructors = EagerInitializedSingleton.class.getDeclaredConstructors();

            for (Constructor constructor : constructors) {

                //Below code will destroy the singleton pattern

                constructor.setAccessible(true);

                instanceTwo = (EagerInitializedSingleton) constructor.newInstance();

                break;

            }

        } catch (Exception e) {

            e.printStackTrace();

        }

        System.out.println(instanceOne.hashCode());

        System.out.println(instanceTwo.hashCode());

    }

}

Iterators are **fail-fast** . i.e. when one thread changes the collection by add / remove operations , while another thread is traversing it through an Iterator using hasNext() or next() method, the iterator fails quickly by throwing ConcurrentModificationException . The fail-fast behavior of iterators can be used only to detect bugs. The Enumerations returned by the methods of classes like Hashtable, Vector are not fail-fast that is achieved by synchronizing the block of code inside the nextElement()method that locks the current Vector object which costs lots of time.

Enumeration                     Iterator

  ----------------                ----------------

  hasMoreElement()                hasNext()

  nextElement()                   next()

  N/A                             remove()

<hibernate-mapping>

   <class name="Employee" table="EMPLOYEE">

      <meta attribute="class-description">

         This class contains the employee detail.

      </meta>

      <id name="id" type="int" column="id">

         <generator class="native"/>

      </id>

      <set name="certificates" cascade="all">

         <key column="employee\_id"/>

         <one-to-many class="Certificate"/>

      </set>

      <property name="firstName" column="first\_name" type="string"/>

      <property name="lastName" column="last\_name" type="string"/>

      <property name="salary" column="salary" type="int"/>

   </class>

   <class name="Certificate" table="CERTIFICATE">

      <meta attribute="class-description">

         This class contains the certificate records.

      </meta>

      <id name="id" type="int" column="id">

         <generator class="native"/>

      </id>

      <property name="name" column="certificate\_name" type="string"/>

   </class>

</hibernate-mapping>

## **Can we Overload or Override static methods in java ?**

Let us first define Overloading and Overriding.

[**Overriding**](http://en.wikipedia.org/wiki/Method_overriding): Overriding is a feature of OOP languages like Java that is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class).  
The implementation to be executed is decided at run-time and decision is made according to the object used for call. Note that signatures of both methods must be same.

[**Overloading**](http://en.wikipedia.org/wiki/Function_overloading): Overloading is also a feature of OOP languages like Java that is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters. Note that in both C++ and Java, [methods cannot be overloaded according to return type.](http://www.geeksforgeeks.org/g-fact-75/)

**Can we overload static methods?**  
The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |

Run on IDE

Output:

Test.foo() called

Test.foo(int) called

**Can we overload methods that differ only by static keyword?**  
We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example. This behaviour is same in C++ (See point 2 of [this](http://www.geeksforgeeks.org/function-overloading-in-c/)).

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public void foo() { // Compiler Error: cannot redefine foo()          System.out.println("Test.foo(int) called ");      }      public static void main(String args[]) {          Test.foo();      }  } |

Run on IDE

Output: Compiler Error, cannot redefine foo()

**Can we Override static methods in java?**  
We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’.  
If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

|  |
| --- |
| /\* Java program to show that if static method is redefined by     a derived class, then it is not overriding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // This method hides display() in Base      public static void display() {           System.out.println("Static or class method from Derived");      }        // This method overrides print() in Base      public void print() {           System.out.println("Non-static or Instance method from Derived");     }  }    // Driver class  public class Test {      public static void main(String args[ ])  {         Base obj1 = new Derived();           // As per overriding rules this should call to class Derive's static         // overridden method. Since static method can not be overridden, it         // calls Base's display()         obj1.display();           // Here overriding works and Derive's print() is called         obj1.print();      }  } |

Run on IDE

Output:

Static or class method from Base

Non-static or Instance method from Derived

Following are some important points for method overriding and static methods in Java.  
**1)** For class (or static) methods, the method according to the type of reference is called, not according to the abject being referred, which means method call is decided at compile time.

**2)** For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time.

**3)** An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors.

|  |
| --- |
| /\* Java program to show that if static methods are redefined by     a derived class, then it is not overriding but hidding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // Static is removed here (Causes Compiler Error)      public void display() {          System.out.println("Non-static method from Derived");      }        // Static is added here (Causes Compiler Error)      public static void print() {          System.out.println("Static method from Derived");     }  } |

Run on IDE

**4)** In a subclass (or Derived Class), we can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods — they are new methods, unique to the subclass.

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed. A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.**java.util.concurrent.Executors** provide implementation of**java.util.concurrent.Executor** interface to create the thread pool in java.  
Let’s write a simple program to explain it’s working.

First we need to have a Runnable class.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | public class WorkerThread implements Runnable { |
| 04 |  |

|  |  |
| --- | --- |
| 05 | private String command; |
| 06 |  |

|  |  |
| --- | --- |
| 07 | public WorkerThread(String s){ |
| 08 | this.command=s; |

|  |  |
| --- | --- |
| 09 | } |
| 10 |  |

|  |  |
| --- | --- |
| 11 | @Override |
| 12 | public void run() { |

|  |  |
| --- | --- |
| 13 | System.out.println(Thread.currentThread().getName()+' Start. Command = '+command); |
| 14 | processCommand(); |

|  |  |
| --- | --- |
| 15 | System.out.println(Thread.currentThread().getName()+' End.'); |
| 16 | } |

|  |  |
| --- | --- |
| 17 |  |
| 18 | private void processCommand() { |

|  |  |
| --- | --- |
| 19 | try { |
| 20 | Thread.sleep(5000); |

|  |  |
| --- | --- |
| 21 | } catch (InterruptedException e) { |
| 22 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 23 | } |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | @Override |

|  |  |
| --- | --- |
| 27 | public String toString(){ |
| 28 | return this.command; |

|  |  |
| --- | --- |
| 29 | } |
| 30 | } |

Here is the test program where we are creating fixed thread pool from Executors framework.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.concurrent.ExecutorService; |
| 04 | import java.util.concurrent.Executors; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class SimpleThreadPool { |

|  |  |
| --- | --- |
| 07 |  |
| 08 | public static void main(String[] args) { |

|  |  |
| --- | --- |
| 09 | ExecutorService executor = Executors.newFixedThreadPool(5); |
| 10 | for (int i = 0; i < 10; i++) { |

|  |  |
| --- | --- |
| 11 | Runnable worker = new WorkerThread('' + i); |
| 12 | executor.execute(worker); |

|  |  |
| --- | --- |
| 13 | } |
| 14 | executor.shutdown(); |

|  |  |
| --- | --- |
| 15 | while (!executor.isTerminated()) { |
| 16 | } |

|  |  |
| --- | --- |
| 17 | System.out.println('Finished all threads'); |
| 18 | } |

|  |  |
| --- | --- |
| 19 |  |
| 20 | } |

In above program, we are creating fixed size thread pool of 5 worker threads. Then we are submitting 10 jobs to this pool, since the pool size is 5, it will start working on 5 jobs and other jobs will be in wait state, as soon as one of the job is finished, another job from the wait queue will be picked up by worker thread and get’s executed.

Here is the output of the above program.

|  |  |
| --- | --- |
| 01 | pool-1-thread-2 Start. Command = 1 |
| 02 | pool-1-thread-4 Start. Command = 3 |

|  |  |
| --- | --- |
| 03 | pool-1-thread-1 Start. Command = 0 |
| 04 | pool-1-thread-3 Start. Command = 2 |

|  |  |
| --- | --- |
| 05 | pool-1-thread-5 Start. Command = 4 |
| 06 | pool-1-thread-4 End. |

|  |  |
| --- | --- |
| 07 | pool-1-thread-5 End. |
| 08 | pool-1-thread-1 End. |

|  |  |
| --- | --- |
| 09 | pool-1-thread-3 End. |
| 10 | pool-1-thread-3 Start. Command = 8 |

|  |  |
| --- | --- |
| 11 | pool-1-thread-2 End. |
| 12 | pool-1-thread-2 Start. Command = 9 |

|  |  |
| --- | --- |
| 13 | pool-1-thread-1 Start. Command = 7 |
| 14 | pool-1-thread-5 Start. Command = 6 |

|  |  |
| --- | --- |
| 15 | pool-1-thread-4 Start. Command = 5 |
| 16 | pool-1-thread-2 End. |

|  |  |
| --- | --- |
| 17 | pool-1-thread-4 End. |
| 18 | pool-1-thread-3 End. |

|  |  |
| --- | --- |
| 19 | pool-1-thread-5 End. |
| 20 | pool-1-thread-1 End. |

|  |  |
| --- | --- |
| 21 | Finished all threads |

The output confirms that there are five threads in the pool named from “pool-1-thread-1? to “pool-1-thread-5? and they are responsible to execute the submitted tasks to the pool.

**Executors** class provide simple implementation of **ExecutorService** using **ThreadPoolExecutor** but ThreadPoolExecutor provides much more feature than that. We can specify the number of threads that will be alive when we create ThreadPoolExecutor instance and we can limit the size of thread pool and create our own **RejectedExecutionHandler**implementation to handle the jobs that can’t fit in the worker queue.

Here is our custom implementation of RejectedExecutionHandler interface.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.concurrent.RejectedExecutionHandler; |
| 04 | import java.util.concurrent.ThreadPoolExecutor; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class RejectedExecutionHandlerImpl implements RejectedExecutionHandler { |

|  |  |
| --- | --- |
| 07 |  |
| 08 | @Override |

|  |  |
| --- | --- |
| 09 | public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) { |
| 10 | System.out.println(r.toString() + ' is rejected'); |

|  |  |
| --- | --- |
| 11 | } |
| 12 |  |

|  |  |
| --- | --- |
| 13 | } |

ThreadPoolExecutor provides several methods using which we can find out the current state of executor, pool size, active thread count and task count. So I have a monitor thread that will print the executor information at certain time interval.

|  |  |
| --- | --- |
| 01 | package com.journaldev.threadpool; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.concurrent.ThreadPoolExecutor; |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public class MyMonitorThread implements Runnable |
| 06 | { |

|  |  |
| --- | --- |
| 07 | private ThreadPoolExecutor executor; |
| 08 |  |

|  |  |
| --- | --- |
| 09 | private int seconds; |
| 10 |  |

|  |  |
| --- | --- |
| 11 | private boolean run=true; |
| 12 |  |

|  |  |
| --- | --- |
| 13 | public MyMonitorThread(ThreadPoolExecutor executor, int delay) |
| 14 | { |

|  |  |
| --- | --- |
| 15 | this.executor = executor; |
| 16 | this.seconds=delay; |

|  |  |
| --- | --- |
| 17 | } |
| 18 |  |

|  |  |
| --- | --- |
| 19 | public void shutdown(){ |
| 20 | this.run=false; |

|  |  |
| --- | --- |
| 21 | } |
| 22 |  |

...