1. When is Singleton is not singleton?

Ans:- In below scenarios Singleton is not Singleton

1. Multiple Threads are invoking Singleclass’s getInstance()

Overcome from this scenario you need to Synchronize the getInstance() LIKE below

public class MySingleton {

private static MySingleton \_instance;

private MySingleton() {

// construct object . . .

}

// For lazy initialization

public static synchronized MySingleton getInstance() {

if (\_instance==null) {

\_instance = new MySingleton();

}

return \_instance;

}

// Remainder of class definition . . .

}

1. While sub-classing the Singleton class - Multiple singletons arising when someone has subclassed your singleton

The uniqueness of the class cannot be imposed as a compile-time constraint on the subclass unless you use a private constructor. If you want to allow subclassing, for example, you might make the constructor protected. A subclass could then expose a public constructor, allowing anyone to make instances. Since an instance of a subclass is an instance of your superclass, you could find multiple instances of the singleton

1. Multiple singletons simultaneously loaded by different class loaders
2. On Multiple VM

When copies of the singleton class run in multiple VMs, an instance is created for each machine.

When two class loaders load a class, you actually have two copies of the class, and each one can have its own singleton instance. That is particularly relevant in servlets running in certain servlet engines (iPlanet for example), where each servlet by default uses its own class loader. Two different servlets accessing a joint singleton will, in fact, get two different objects.

1. Sinlgeton class implementing Seriazable interface - Copies of a singleton object that has undergone serialization and deserialization . If you have a serialized object and deserialize it twice in different ObjectOutputStreams, or with calls ObjectOutputStream.reset() between deserializations, you get two distinct objects, not two references to the same object.

It is design decision to have Singleton class Serialized. Serialization breaks Singleton class property so it is recommended not to have class as Serialized

1. Synchronization in Singleton class?

Refer point one (i) of above question answer

1. Synchronization class implements serialization, clonable how to handle this

Refer point one (v) of above question answer for clonable – when u implement clonable interface u need to override clone method , and return reference of the instance created prior

Use of Generics other than Collection- Arrays doesn't support Generics in Java so you can not create Arrays like T[] which makes gentrifying an existing class hard if you are using arrays. Though there are work around which requires a cast from Object[] to T[] which comes with risk of unchecked cast and warning. For this reason it's better to use Collections classes like ArrayList and HashMap over array. by the way those classes are also implemented on top of array in Java but JDK handles there type-safety by effectively using generics. here is an example of casting Object array to generic array in Java :

/\*\*  
 \* Generics and Array doesn't gel very well, Java doesn’t allow Generics array like E[]   
 \* @author Javin Paul  
 \*/  
**public** **class** GenericVsArray {  
    
    **public** **static** **void** main(**String** args[]){  
      Holder<**Integer**> numbers = **new** Holder<**Integer**>(10);  
      numbers.add(101);  
      **System**.out.println("Get: " + numbers.get(0));  
    }  
  
    
}  
  
/\*\*  
 \* Generic Holder for holding contents of different object type  
 \* Generic in Java eliminates casting required while calling get(index) from client code  
 \* @param <T>  
 \*/  
**class** Holder<T>{  
    **private** T[] contents;  
    **private** **int** index = 0;  
    **public** Holder(**int** size){  
        *//contents = new T[size]; //compiler error - generic array creation*  
        contents = (T[]) **new** **Object**[size]; *//workaround - casting Object[] to generic Type*  
    }  
    
    **public** **void** add(T content){  
        contents[index] = content;  
    }  
    
    **public** T get(**int** index){  
        **return** contents[index];  
    }  
}

Casting code may generate warning about "unsafe cast" which can be suppressed by using annotation @SuppressWarnings("unchecked") with proper comment that why it will not compromise type-safety. This is also one of the Java Generics best practices suggested in all time classic book [Effective Java by Joshua Bloch](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fwww.amazon.com%2fdp%2f0321356683%2f%3ftag%3djavamysqlanta-20).

**Generics in Java – Benefits and advantages**

Generics adds lot of value into Java programming language, here are some of important benefits of using Generics in Java:

**Type-safety**

Most important advantage of Generics in Java is **type-safety**. Collections prior to JDK1.5 are not type-safe because they accept Object type argument which allows them to catch all type of objects instead of only required type of object. For example if you want to create an ArrayList of Stocks and you don't want that ArrayList also contain any other asset class you can use generics feature of java to create a **type-safe collection**. Here is an example of using Generics to create a type-safe ArrayList

[**ArrayList**](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2011%2f05%2fexample-of-arraylist-in-java-tutorial.html)<Stocks> stockList = **new** **ArrayList**<Stocks>();

stockList.add(“coins”); //compiler error , String not allowed

Compiler will guaranteed that only Stock object will be inserted in stockList and will [throw](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2012%2f02%2fdifference-between-throw-and-throws-in.html) compiler error if you try to insert different type of Object.

**No Casting**

With Generics you don’t need to cast object , Generics will automatically do that for you. For example here is the code for adding and retrieving an element in List with and without Generics in Java:

**List**  items = **new** **ArrayList**();  
items.add("chocolates");  
**String** item = (**String**) items.get(0)  
  
**List**<**String**> items = **new** **ArrayList**();  
items.add("biscuits");  
**String** item = items.get(0) *//no cast required*

Since no cast required, result is clear and robust code.

**No ClassCastException**

With Generics compiler ensures that correct types are added into Java collection classes and no cast is required while retrieving element, So there is no risk of ClassCastException at [runtime](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2012%2f03%2fwhat-is-static-and-dynamic-binding-in.html).

## **Generics in Java – Important points**

Some important feature of Generics in Java worth remembering:

1) One limitation of Generics in Java is that it can not be applied to primitive type, for example you can not create pass primitives in angle bracket that will result in compilation error, for Example ArrayList<int> will result in compilation error, This is little counter-intuitive that [why auto-boxing can not covert int to Integer](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2012%2f07%2fauto-boxing-and-unboxing-in-java-be.html). If you try same thing with our Generic Holder class you will get following compilation error:

Holder<**int**> numbers = **new** Holder<**int**>(10); *//compiler error - unexpected type required: reference found:int*

2) Generics in Java eliminates ClassCastException while retrieving objects from Collection, Remember prior to JDK1.5 if you retrieve objects from Collection you first check for a particular type and then cast, no need to do it now.

**ArrayList**<Stocks> stockList = **new** **ArrayList**<StockList>();  
Stock sony = **new** Stock("Sony","6758.T");  
stockList.add(sony);  
Stock retreivedStock = stockList.get(sony); *//no cast requires – automatic casting by compiler*

3) A [parametrized class in Java](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2011%2f10%2fclass-in-java-programming-general.html) use formal type parameters to retrieve Type information when instance of parametrized class gets created. In below example of generics class in Java <K,V> are formal parameters.

**interface** Cache <K,V>{  
        **public** V get();  
        **public** V put(K key, V value);  
}

As per convention followed on Generics version of [Java Collection](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2011%2f11%2fcollection-interview-questions-answers.html) package we can use <K,V> for key and value type parameters.

4) Generics are often related to **Templates in C++**, though Unlike "Template" in C++, which creates a new type for each specific parametrized type, parametrized class in Java is only compiled once and more importantly there is just one single class file which is used to create instances for all the specific types.

5) Generics in Java can not only apply on [Java Classes](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.com%2f2011%2f10%2fclass-in-java-programming-general.html) but also on methods, so you can write your own generics methods in Java as shown on Rules of Generics in Java section, here is another example of parametrized method from Java collection package.

boolean add(E o){}

Here E will be replaced by actual type parameter when this method will get called.

6) Another worth noting feature of Generics in Java is its ability to limit Types parameters, for example in parametric declaration of Holder<T extends Closeable>, type parameter list <T extends Closeable> requires that actual parameter T must be either Closeable or sub-type of Closeable. This is called [bounded type parameters in Generics](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2012%2f04%2fwhat-is-bounded-and-unbounded-wildcards.html) . this kind of declaration allows you to call method of Closeable interface without casting type parameter into Closeable. read more about these type parameters in bounded and unbounded wildcards in Generics.

7) **Type inference** : Generics in Java does not support type inference while calling constructor or creating instance of Generic Types until JDK7, In Java 7 along with [Automatic resource management](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2011%2f09%2farm-automatic-resource-management-in.html) and [String in Switch](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2011%2f08%2fstring-switch-case-jdk7-example.html)  also added a new operator called Diamond operator and denoted by <> which facilitate type inference while creating instance of Generics classes. this helps to reduce redundancy and clutter. here is an example of Diamond operator in Java7 code:

*//prior to JDK 7*  
[**HashMap**](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.com%2f2011%2f02%2fhow-hashmap-works-in-java.html)<**String**, **Set**<**Integer**>> contacts = **new** **HashMap**<**String**, **Set**<**Integer**>>()  
  
*//JDK 7 diamond operator*  
**HashMap**<**String**, **Set**<**Integer**>> contacts = **new** **HashMap**<>()

code with diamond operator is much cleaner than previous one.

On related note Generics in Java supports type inference while calling Generic methods and this feature can be used to create in combination of [Factory design pattern in Java](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2011%2f12%2ffactory-design-pattern-java-example.html) to create static factory method corresponding to each constructors. for example

*//type inference in generic method*  
**public** **static** <K,V> **HashMap**<K,V> newContacts() {  
   **return** **new** **HashMap**<K,V>();  
}

so we can replace call to constructor with this static factory method as shown below :

**HashMap**<**String**, **Set**<**Integer**>> contacts = newContacts();

this can be used as alternative to diamond operator in Java 5 or 6.

## **Section for absolute beginners on Generics in Java**

If you are absolute beginners in generics those angle bracket "<>" may look strange and unreadable to you. Though is not a complete tutorial on Java Generics and I would suggest you to read Java docs on Generics I will try to give at least some basic idea of generics in Java to get you going. Remember Generics in java are introduced to enforce type-safety especially on collection classes of java which holds type of Object e.g. ArrayList, HashMap.

Type-safety means compiler will verify type of class during compile time and throw compiler error if it found  improper type. For example if an ArrayList of Gold contains Silver compiler will throw error.

ArrayList<Gold> goldList = new ArrayList<Gold>();

<Gold> tells compiler that this ArrayList must contain only Gold.

Generics can also be used to write parametric classes like Cache<Key, Value> on which type of Key and Value can be specified while creating objects.

Parameters used to write code is called **"formal type parameters"** and parameters which passed while creating instance of a generic class in java is called **"actual type parameters".** For example in our generic cache (below) <K, V> are formal parameter while new LRUCache<String, Integer>() will be **actual parameters.**

**Generics wild cards Example in Java**

There are generally two kinds of wild-cards in Generics, Bounded and unbounded. Bounded wildcards can be written in two ways to denote upper bound and lower bound. <?> is called unbounded wildcards because it can accept any Type while <? extends T> and <? super T> are bounded wildcards. To know more about them see my post [Bounded vs Unbounded wildcards in Generics](https://mail.cognizant.com/owa/redir.aspx?C=5oxAoubFLE22_6Ci9O-qPiwwkPxrwNAIgBtCGIRXNXQ-mk2IFhgzqHo22bKfpSZNnSL67AlQAk0.&URL=http%3a%2f%2fjavarevisited.blogspot.sg%2f2012%2f04%2fwhat-is-bounded-and-unbounded-wildcards.html) . Now let’s see example of different wildcards in Generics:

**<?>**

*"*?" denotes any unknown type, It can represent any Type at in code for. Use this wild card if you are not sure about Type. for example if you want to have a ArrayList which can work with any type than declare it as  "ArrayList<?> unknownList" and it can be assigned to any type of ArrayList as shown in following example of generics in Java:

**ArrayList**<?> unknownList = **new** **ArrayList**<**Number**>();  
unknownList = **new** **ArrayList**<**Float**>();

**<? extends T>**

This is little restrictive than previous one it will allow All Types which are either "T" or extends T means subclass of T. for example List<? extends Number> can hold List<Number> or List<Integer>

**ArrayList**<? **extends** **Number**> numberList = **new** **ArrayList**<**Number**>();  
numberList = **new** **ArrayList**<**Integer**>();  
numberList = **new** **ArrayList**<**Float**>();

**<T super ?>**

This is just opposite of previous one, It will allow T and super classes of T, e.g. List<? super Integer>can hold List<Integer> or List<Number>.

**ArrayList**<? **super** **Integer**> numberList = **new** **ArrayList**<**Number**>();  
numberList = **new** **ArrayList**<**Integer**>();  
numberList = **new** **ArrayList**<**Float**>(); *//compilation error*

1. [difference between Struts 1.x and Struts 2.x](http://stackoverflow.com/questions/793291/difference-between-struts-1-x-and-struts-2-x)

Struts2 works with POJOs, no more ActionForms. All attributes from the requests are populated in "standard" Java classes.

Struts2 actions are not coupled with the Servlet API, they do not receive HttpServletRequest and HttpServletResponse as parameters. Also, they are stateful, (Struts 1 actions were stateless).

Struts2 have a lot of enhancements like the Interceptor API, plus lots of integrations.

Configuration is totally different.

1. What is Interceptor in struts

Interceptors are conceptually the same as servlet filters or the JDKs Proxy class. Interceptors allow for crosscutting functionality to be implemented separately from the action as well as the framework. You can achieve the following using interceptors:

* Providing preprocessing logic before the action is called.
* Providing postprocessing logic after the action is called.
* Catching exceptions so that alternate processing can be performed.

Many of the features provided in the Struts2 framework are implemented using interceptors

Create Interceptor Class:

package com.tutorialspoint.struts2;

import java.util.\*;

import com.opensymphony.xwork2.ActionInvocation;

import com.opensymphony.xwork2.interceptor.AbstractInterceptor;

public class MyInterceptor extends AbstractInterceptor {

public String intercept(ActionInvocation invocation)throws Exception{

/\* let us do some pre-processing \*/

String output = "Pre-Processing";

System.out.println(output);

/\* let us call action or next interceptor \*/

String result = invocation.invoke();

/\* let us do some post-processing \*/

output = "Post-Processing";

System.out.println(output);

return result;

}

}

## Create Action Class:

package com.tutorialspoint.struts2;

import com.opensymphony.xwork2.ActionSupport;

public class HelloWorldAction extends ActionSupport{

private String name;

public String execute() throws Exception {

System.out.println("Inside action....");

return "success";

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

## Create a View

<%@ page contentType="text/html; charset=UTF-8" %>

<%@ taglib prefix="s" uri="/struts-tags" %>

<html>

<head>

<title>Hello World</title>

</head>

<body>

Hello World, <s:property value="name"/>

</body>

</html>

## Create main page:

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib prefix="s" uri="/struts-tags"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<title>Hello World</title>

</head>

<body>

<h1>Hello World From Struts2</h1>

<form action="hello">

<label for="name">Please enter your name</label><br/>

<input type="text" name="name"/>

<input type="submit" value="Say Hello"/>

</form>

</body>

</html>

## Configuration Files

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name="struts.devMode" value="true" />

<package name="helloworld" extends="struts-default">

<interceptors>

<interceptor name="myinterceptor"

class="com.tutorialspoint.struts2.MyInterceptor" />

</interceptors>

<action name="hello"

class="com.tutorialspoint.struts2.HelloWorldAction"

method="execute">

<interceptor-ref name="params"/>

<interceptor-ref name="myinterceptor" />

<result name="success">/HelloWorld.jsp</result>

</action>

</package>

</struts>

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

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

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>Struts 2</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<filter>

<filter-name>struts2</filter-name>

<filter-class>

org.apache.struts2.dispatcher.FilterDispatcher

</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

1. Id in hibernate hbm file?

The following are the list of main generators we are using in the hibernate framework

* *assigned -*

*This generator supports in all the databases*

*This is the default generator class used by the hibernate, if we do not specify <generator –> element under id element then hibernate by default assumes it as “assigned”*

*If generator class is assigned, then the programmer is responsible for assigning the primary key value to object which is going to save into the database*

* increment

This generator supports in all the databases, database independent

This generator is used for generating the id value for the new record by using the formula

Max of id value in Database + 1

if we manually assigned the value for primary key for an object, then hibernate doesn’t considers that value and uses max value of id in database + 1 concept only

If there is no record initially in the database, then for the first time this will saves primary key value as 1, as…

* sequence

Not has the support with MySql

This generator class is database dependent it means, we cannot use this generator class for all the database, we should know whether the database supports sequence or not before we are working with it

while inserting a new record in a database, hibernate gets next value from the sequence under assigns that value for the new record

If programmer has created a sequence in the database then that sequence name should be passed as the generator

<id name="productId" column="pid">

<generator>

<param name="sequence">MySequence</param>

</genetator>

</id>

<id name="productId" column="pid"><generator><param name="sequence">MySequence</param></genetator></id>

If the programmer has not passed any sequence name, then hibernate creates its own sequence with name “Hibernate-Sequence” and gets next value from that sequence, and than assigns that id value for new record

But remember, if hibernate want’s to create its own sequence, in hibernate configuration file, hbm2ddl.auto property must be set enabled

* identity

This is database dependent, actually its not working in oracle

In this case (identity generator) the id value is generated by the database, but not by the hibernate, but in case of increment hibernate will take over this

this identity generator doesn’t needs any parameters to pass

this identity generator is similar to increment generator, but the difference was increment generator is database independent and hibernate uses a select operation for selecting max of id before inserting new record

But in case of identity, no select operation will be generated in order to insert an id value for new record by the hibernate

<id name="productid" column="pid">

<generator class="......."/>

</id>

<id name="productid" column="pid"><generator class="......."/></id>

As this is not working in Oracle, if you would like to check this in MySql you must change the configuration file as…….

class: com.mysql.jdbc.Driver

url: jdbc:mysql://www.java4s.com:3306/test (test is default database)

user: root (default)

pass: (default)

dialet: org.hibernate.dialet.MySQLDialet

* hilo
* native
* foregin
* uuid.hex
* uuid.string

1. Multithreading in Servlet

A Java servlet container / web server is typically multithreaded. That means, that multiple requests to the same servlet may be executed at the same time. Therefore, you need to take concurrency into consideration when you implement your servlet.

I am not going to describe Java concurrency in great detail here. If you are interested in this topic, read my tutorial on [Java Concurrency](http://tutorials.jenkov.com/java-concurrency/index.html).

To make sure that a servlet is thread safe, there are a few basic rules of thumb you must follow:

1. Your servlet service() method should not access any member variables, unless these member variables are thread safe themselves.
2. Your servlet service() should not reassign member variables, as this may affect other threads executing inside the service() method. If you really, really need to reassign a member variable, make sure this is done inside a synchronized block.
3. Rule 1 and 2 also counts for static variables.
4. Local variables are always thread safe. Keep in mind though, that the object a local variable points to, may not be so. If the object was instantiated inside the method, and never escapes, there will be no problem. On the other hand, a local variable pointing to some shared object, may still cause problems. Just because you assign a shared object to a local reference, does not mean that object automatically becomes thread safe.

The request and response objects are of course thread safe to use. A new instance of these are created for every request into your servlet, and thus for every thread executing in your servlet.

1. One – Many relationship in hibernate

*File : Stock.hbm.xml*

**<?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.mkyong.stock.Stock" table="stock" catalog="mkyongdb"**>**

**<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>**

**<property** name="stockName" type="string"**>**

**<column** name="STOCK\_NAME" length="20" not-null="true" unique="true" **/>**

**</property>**

**<set** name="stockDailyRecords" table="stock\_daily\_record"

inverse="true" lazy="true" fetch="select"**>**

**<key>**

**<column** name="STOCK\_ID" not-null="true" **/>**

**</key>**

**<one-to-many** class="com.mkyong.stock.StockDailyRecord" **/>**

**</set>**

**</class>**

**</hibernate-mapping>**

*File : StockDailyRecord.hbm.xml*

**<?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.mkyong.stock.StockDailyRecord" table="stock\_daily\_record"

catalog="mkyongdb"**>**

**<id** name="recordId" type="java.lang.Integer"**>**

**<column** name="RECORD\_ID" **/>**

**<generator** class="identity" **/>**

**</id>**

**<many-to-one** name="stock" class="com.mkyong.stock.Stock" fetch="select"**>**

**<column** name="STOCK\_ID" not-null="true" **/>**

**</many-to-one>**

**<property** name="priceOpen" type="java.lang.Float"**>**

**<column** name="PRICE\_OPEN" precision="6" **/>**

**</property>**

**<property** name="priceClose" type="java.lang.Float"**>**

**<column** name="PRICE\_CLOSE" precision="6" **/>**

**</property>**

**<property** name="priceChange" type="java.lang.Float"**>**

**<column** name="PRICE\_CHANGE" precision="6" **/>**

**</property>**

**<property** name="volume" type="java.lang.Long"**>**

**<column** name="VOLUME" **/>**

**</property>**

**<property** name="date" type="date"**>**

**<column** name="DATE" length="10" not-null="true" unique="true" **/>**

**</property>**

**</class>**

**</hibernate-mapping>**

**. Hibernate Configuration File**

**<?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>**

**<property** name="hibernate.connection.driver\_class"**>**com.mysql.jdbc.Driver**</property>**

**<property** name="hibernate.connection.url"**>**jdbc:mysql://localhost:3306/mkyongdb**</property>**

**<property** name="hibernate.connection.username"**>**root**</property>**

**<property** name="hibernate.connection.password"**>**password**</property>**

**<property** name="hibernate.dialect"**>**org.hibernate.dialect.MySQLDialect**</property>**

**<property** name="show\_sql"**>**true**</property>**

**<property** name="format\_sql"**>**true**</property>**

**<mapping** resource="com/mkyong/stock/Stock.hbm.xml" **/>**

**<mapping** resource="com/mkyong/stock/StockDailyRecord.hbm.xml" **/>**

**</session-factory>**

**</hibernate-configuration>**