Q: How to remove duplicate rows from table in Oracle?

Ans:

DELETE FROM EMP\_TEMP\_DUPS

WHERE rowid not in

(SELECT MIN (rowid)

FROM EMP\_TEMP\_DUPS

GROUP BY id,firstname,lastname,age,salary);

Q: How to identify whether the linked list has loop?

private static boolean isCircularLinkedList(LinkedList<Integer> linkedList) {

boolean isCircular = Boolean.FALSE;

Set<Integer> circular = new HashSet<>();

for (Integer integer : linkedList) {

if (circular.contains(integer)) {

System.out.println("Circular Linked List: " + integer);

isCircular = Boolean.TRUE;

break;

} else {

circular.add(integer);

}

}

return isCircular;

}

Q: when should I use JTA transaction manager and when JPA, and what benefit and disadvantages does they have?

If you want to delegate managed transactions to your Application Server and handle complex transactions across multiple resources you need to use the JtaTransactionManager,

The Spring Framework gives you the choice of when to scale your application to a fully loaded application server. Gone are the days when the only alternative to using EJB CMT or JTA was to write code with local transactions such as those on JDBC connections, and face a hefty rework if you need that code to run within global, container-managed transactions. With the Spring Framework, only some of the bean definitions in your configuration file, rather than your code, need to change

**Spring Framework transaction management**

### **Global transactions**

Global transactions enable you to work with multiple transactional resources, typically relational databases and message queues. The application server manages global transactions through the JTA, which is a cumbersome API to use (partly due to its exception model). Furthermore, a JTA UserTransaction normally needs to be sourced from JNDI, meaning that you *also* need to use JNDI in order to use JTA. Obviously the use of global transactions would limit any potential reuse of application code, as JTA is normally only available in an application server environment.

### Local transactions

Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have significant disadvantages: they cannot work across multiple transactional resources. For example, code that manages transactions using a JDBC connection cannot run within a global JTA transaction. Because the application server is not involved in transaction management, it cannot help ensure correctness across multiple resources. (It is worth noting that most applications use a single transaction resource.) Another downside is that local transactions are invasive to the programming model.

The TransactionDefinition interface specifies:

* *Isolation*: The degree to which this transaction is isolated from the work of other transactions. For example, can this transaction see uncommitted writes from other transactions?
* *Propagation*: Typically, all code executed within a transaction scope will run in that transaction. However, you have the option of specifying the behavior in the event that a transactional method is executed when a transaction context already exists. For example, code can continue running in the existing transaction (the common case); or the existing transaction can be suspended and a new transaction created. *Spring offers all of the transaction propagation options familiar from EJB CMT*. To read about the semantics of transaction propagation in Spring, see [Section 11.5.7, “Transaction propagation”](https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/transaction.html#tx-propagation).
* *Timeout*: How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.
* *Read-only status*: A read-only transaction can be used when your code reads but does not modify data. Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.

Q: How can you display top two employee salaries group by department in oracle?

select

d.Salary

,d.Department

from

(

select

r.Salary

,r.Department

,row\_number() over(

partition by r.Department

order by r.Salary desc) as RowNumber

from HumanResources as r

) as d

where d.RowNumber < 3

# [**Finding duplicate values in a SQL table**](https://stackoverflow.com/questions/2594829/finding-duplicate-values-in-a-sql-table)

ID NAME EMAIL

1 John asd@asd.com

2 Sam asd@asd.com

3 Tom asd@asd.com

4 Bob bob@asd.com

5 Tom asd@asd.com

Ans:

SELECT

name, email, COUNT(\*)

FROM

users

GROUP BY

name, email

HAVING

COUNT(\*) > 1

Q: How to find top three highest salary in emp table in oracle?

SELECT \*FROM

(

SELECT \*FROM emp

ORDER BY Salary desc

)

WHERE rownum <= 3

ORDER BY Salary ;

SELECT \* FROM

(

SELECT EMPLOYEE, LAST\_NAME, SALARY,

RANK() OVER (ORDER BY SALARY DESC) EMPRANK

FROM emp

)

WHERE emprank <= 3;

# Q: [Highest Salary in each department](https://stackoverflow.com/questions/8477040/highest-salary-in-each-department)

Table Name EmpDetails

DeptID EmpName Salary

Engg Sam 1000

Engg Smith 2000

HR Denis 1500

HR Danny 3000

IT David 2000

IT John 3000

Ans:

SELECT DeptID, MAX(Salary) FROM EmpDetails GROUP BY Dept

# Q:[Differences between session vs session factory - Hibernate?](https://stackoverflow.com/questions/22470968/differences-between-session-vs-session-factory-hibernate)

SessionFactory is Hibernate’s concept of a single datastore and is threadsafe so that many threads can access it concurrently and request for sessions and immutable cache of compiled mappings for a single database.

No, Session is not Thread Safe. A Session is a light weight and a non-threadsafe object (No, you cannot share it between threads) that represents a single unit-of-work with the database. Sessions are opened by a SessionFactory and then are closed when all work is complete. Session is the primary interface for the persistence service. A session obtains a database connection lazily (i.e. only when required)

Q: Is Hibernate's session thread safe?

t is not intended that implementors be threadsafe. Instead each thread/transaction should obtain its own instance from a SessionFactory.

Even with this in mind, your behaviour might still not be what you expect, because transactions come into play. You will have to set a proper [transaction isolation level](http://en.wikipedia.org/wiki/Isolation_(database_systems)). See the [configuration guide](http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html), hibernate.connection.isolation property.

It depends on how you are creating a session.

Session can be created in two ways in hibernate.

1. getCurrentSession()

Yes. It offers thread safety as it'll ensure that it'll create a session for each thread if session not exist. transaction and automatic session closing is attached to this.

1. openSession()

It's not thread safe. developer manually needs to manage transactions and session flush and close operations.

**Spring Boot**:

Spring Boot is a lightweight framework that takes most of the work out of configuring Spring-based applications.

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run." We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss.

### **Starters**

Starters are a big part of the magic of Spring Boot, used to limit the amount of manual dependency configuration that you have to do.

All starters use the naming convention: spring-boot-starter-XYZ, where XYZ is the type of application you want to build. Here are some popular Spring Boot starters:

* spring-boot-starter-**web** is used to build RESTful web services using Spring MVC and Tomcat as the embedded application container.
* spring-boot-starter-**jersey** is an alternative to spring-boot-starter-web that uses Apache Jersey rather than Spring MVC.
* spring-boot-starter-**jdbc** is used for JDBC connection pooling. It's based on Tomcat's JDBC connection-pool implementation.

### **Auto-configuration**

If you let it, Spring Boot will use its @EnableAutoConfiguration annotation to automatically configure your application. Auto-configuration is based on the JARS in your classpath and how you've defined your beans:

**Take a look**: Fire up your Spring Boot application with the --debug option and an auto-configuration report will be generated to the console.

* Spring Boot uses the JARs you have specified to be present in theCLASSPATH to form an opinion about how to configure certain automatic behavior. For example, if you have the H2 database JAR in your classpath and have configured no other DataSource beans, then your application will be automatically configured with an in-memory database.

**Q: Difference between @Primary vs @Autowired with @Qualifier annotations**

Read [@Primary](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Primary.html) is the "default".

If a bean has @Autowired without any @Qualifier, and multiple beans of the type exist, the candidate bean marked @Primary will be chosen, i.e. it is the default selection when no other information is available, i.e. when @Qualifier is missing.

A good use case is that initially you only had one bean of the type, so none of the code used @Qualifier.

When you then add another bean, you then also add @Qualifier to both the old and the new bean, so any @Autowired can choose which one it wants.

By also adding @Primary to the old original bean, you don't have to add @Qualifier to all the existing @Autowired. They are "grandfathered" in, so to speak.

@Primary is also good if e.g. 95% of @Autowired wants a particular bean. That way, only the @Autowired want the other bean(s) need to specify @Qualifier. That way, you have primary beans that all autowired wants, and @Qualifier is only used to request an "alternate" bean.

**Q: What will happen if we define same bean(Scope=”Singleton”) twice in Spring applicationContext.xml with different id?**

Ans: in spite of setting scope is “Singleton”, spring will create two instances.

**applicationContext.xml:**

<bean id=*"helloBean"* class=*"com.spring.core.HelloWorld"* scope=*"singleton"*>

<property name=*"name"* value=*"Srihith"* />

</bean>

<bean id=*"helloBean2"* class=*"com.spring.core.HelloWorld"* scope=*"singleton"*>

<property name=*"name"* value=*"Prasad"* />

</bean>

ApplicationContext context = **new** ClassPathXmlApplicationContext(

"SpringBeans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloBean");

obj.printHello();

System.***out***.println("Obj=-->"+obj);

HelloWorld obj2 = (HelloWorld) context.getBean("helloBean2");

System.***out***.println("Obj2=-->"+obj2);

obj2.printHello();

Output:

Spring 3: Hello ! Srihith

Obj=-->com.spring.core.HelloWorld@2d209079

Obj2=-->com.spring.core.HelloWorld@6bdf28bb

Spring 3: Hello! Prasad

**Q: what will happen if the constructor is private in spring bean class> will spring create an instance?**

Ans: yes spring will create an instance.

Ex:

**package** com.spring.core;

/\*\*

\* Spring bean

\*

\*/

**public** **class** HelloWorld {

**private** String name;

**private** HelloWorld() {

System.***out***.println("Private Constructor...>");

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **void** printHello() {

System.***out***.println("Spring 3 : Hello ! " + name);

}

}

**package** com.spring.core;

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

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

**public** **class** App {

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

ApplicationContext context = **new** ClassPathXmlApplicationContext(

"SpringBeans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloBean");

obj.printHello();

System.***out***.println("Obj=-->"+obj);

HelloWorld obj2 = (HelloWorld) context.getBean("helloBean2");

System.***out***.println("Obj2=-->"+obj2);

obj2.printHello();

}

}

Output:

Private Constructor...>

Private Constructor...>

Spring 3 : Hello ! Srihith

Obj=-->com.spring.core.HelloWorld@2d209079

Obj2=-->com.spring.core.HelloWorld@6bdf28bb

Spring 3 : Hello ! Prasad

Q: **BeanFactory vs ApplicationContext?**

**Bean Factory**

Bean instantiation/wiring

**Application Context**

Bean instantiation/wiring

Automatic BeanPostProcessor registration

Automatic BeanFactoryPostProcessor registration

Convenient MessageSource access (for i18n)

ApplicationEvent publication

So if you need any of the points presented on the Application Context side, you should use ApplicationContext.

Q: **What is Automatic registration of BeanPostProcessor?**

You shouldn't actually care unless you have a very specific use case. In certain environments, you may want to turn off some additional features that ApplicationContext has over BeanFactory. These are summarized [in this table](http://docs.spring.io/spring/docs/4.0.2.RELEASE/spring-framework-reference/html/beans.html#context-introduction-ctx-vs-beanfactory)

As for "Automatic registration of BeanPostProcessor", the ApplicationContext will detect beans that implement BeanPostProcessor and will use them to enrich the context while it is loaded. This is a very powerful feature of Spring, check [Container extension points](http://docs.spring.io/spring/docs/4.0.2.RELEASE/spring-framework-reference/html/beans.html#beans-factory-extension)

/\*\*

\*

\*/

**package** com.spring.core;

**import** org.springframework.beans.BeansException;

**import** org.springframework.beans.factory.config.BeanPostProcessor;

**public** **class** CustomBeanPostProcessor **implements** BeanPostProcessor {

**public** Object postProcessAfterInitialization(Object arg0, String arg1) **throws** BeansException {

System.***out***.println("postProcessAfterInitialization arg0:"+arg0+", arg1:"+arg1);

**return** arg0;

}

**public** Object postProcessBeforeInitialization(Object arg0, String arg1) **throws** BeansException {

System.***out***.println("postProcessBeforeInitialization arg0:"+arg0+", arg1:"+arg1);

**return** arg0;

}

}

**Output:**

Private Constructor...>

postProcessBeforeInitialization arg0:com.spring.core.HelloWorld@54c562f7, arg1:helloBean

postProcessAfterInitialization arg0:com.spring.core.HelloWorld@54c562f7, arg1:helloBean

Private Constructor...>

postProcessBeforeInitialization arg0:com.spring.core.HelloWorld@318ba8c8, arg1:helloBean2

postProcessAfterInitialization arg0:com.spring.core.HelloWorld@318ba8c8, arg1:helloBean2

Q: Is Singleton Spring bean is thread-safe?

Ans: no. Ideally spring beans do not have state.

Q: Java 7 new Features?

1. Try-with-resources statement
2. Underscores in numeric literals
3. Strings in switch
4. Binary literals
5. Multiple exception catching
6. Improved Type Inference for Generic Instance Creation
7. SafeVarargs

## Java SE 7 [Features and Enhancements](http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html) from JDK 7 Release Notes

This is the Java 7 new features summary from the [OpenJDK 7 features page](http://openjdk.java.net/projects/jdk7/features/):

vm JSR 292: Support for dynamically-typed languages (InvokeDynamic)

Strict class-file checking

lang JSR 334: Small language enhancements (Project Coin)

core Upgrade class-loader architecture

Method to close a URLClassLoader

Concurrency and collections updates (jsr166y)

i18n Unicode 6.0

Locale enhancement

Separate user locale and user-interface locale

ionet JSR 203: More new I/O APIs for the Java platform (NIO.2)

NIO.2 filesystem provider for zip/jar archives

SCTP (Stream Control Transmission Protocol)

SDP (Sockets Direct Protocol)

Use the Windows Vista IPv6 stack

TLS 1.2

sec Elliptic-curve cryptography (ECC)

jdbc JDBC 4.1

client XRender pipeline for Java 2D

Create new platform APIs for 6u10 graphics features

Nimbus look-and-feel for Swing

Swing JLayer component

Gervill sound synthesizer [NEW]

web Update the XML stack

mgmt Enhanced MBeans [UPDATED]

## Code examples for new features in Java 1.7

### **Try-with-resources statement**

this:

BufferedReader br = new BufferedReader(new FileReader(path));

try {

return br.readLine();

} finally {

br.close();

}

becomes:

try (BufferedReader br = new BufferedReader(new FileReader(path)) {

return br.readLine();

}

You can declare more than one resource to close:

try (

InputStream in = new FileInputStream(src);

OutputStream out = new FileOutputStream(dest))

{

// code

}

### **Underscores in numeric literals**

int one\_million = 1\_000\_000;

### **Strings in switch**

String s = ...

switch(s) {

case "quux":

processQuux(s);

// fall-through

case "foo":

case "bar":

processFooOrBar(s);

break;

case "baz":

processBaz(s);

// fall-through

default:

processDefault(s);

break;

}

### **Binary literals**

int binary = 0b1001\_1001;

### **Improved Type Inference for Generic Instance Creation**

Map<String, List<String>> anagrams = new HashMap<String, List<String>>();

becomes:

Map<String, List<String>> anagrams = new HashMap<>();

### **Multiple exception catching**

this:

} catch (FirstException ex) {

logger.error(ex);

throw ex;

} catch (SecondException ex) {

logger.error(ex);

throw ex;

}

becomes:

} catch (FirstException | SecondException ex) {

logger.error(ex);

throw ex;

}

### **SafeVarargs**

this:

@SuppressWarnings({"unchecked", "varargs"})

public static void printAll(List<String>... lists){

for(List<String> list : lists){

System.out.println(list);

}

}

becomes:

@SafeVarargs

public static void printAll(List<String>... lists){

for(List<String> list : lists){

System.out.println(list);

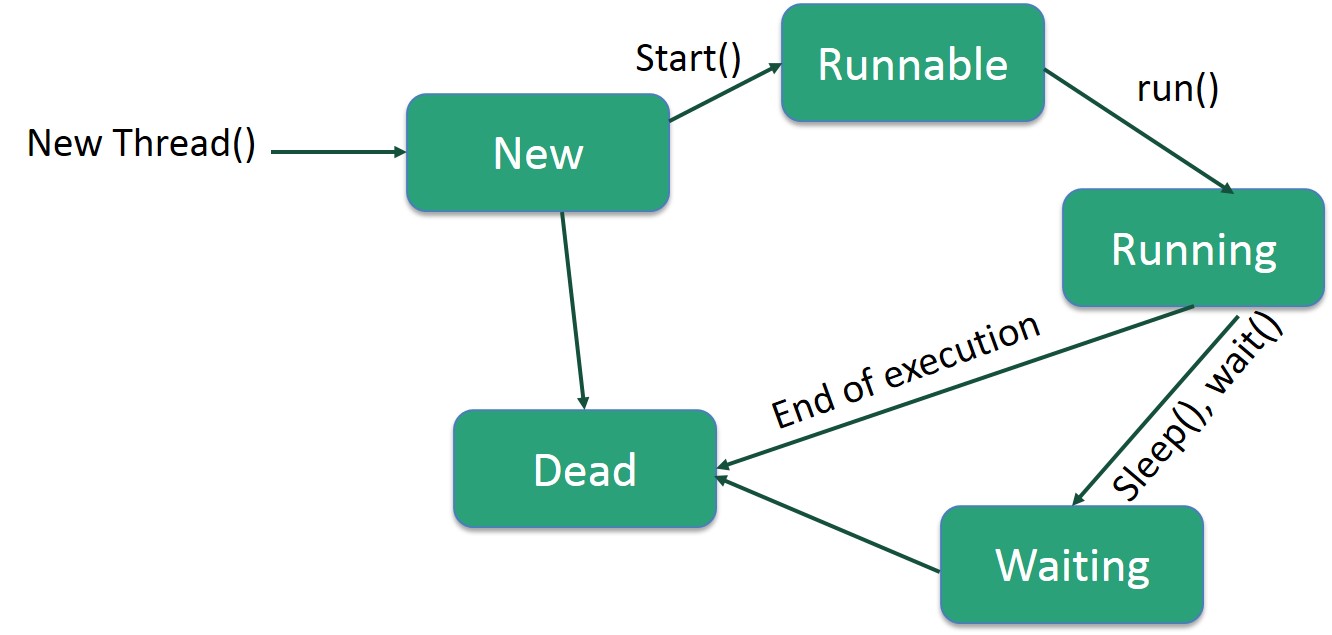
}

}

Q: Java 8 new features?

1. forEach() method in Iterable interface.
2. default and static methods in Interfaces.
3. Functional Interfaces and Lambda Expressions.
4. Java Stream API for Bulk Data Operations on Collections.
5. Java Time API.
6. Collection API improvements.
7. Concurrency API improvements.
8. Java IO improvements.

Thread Life Cycle:



Java Thread Above-mentioned stages are explained here:

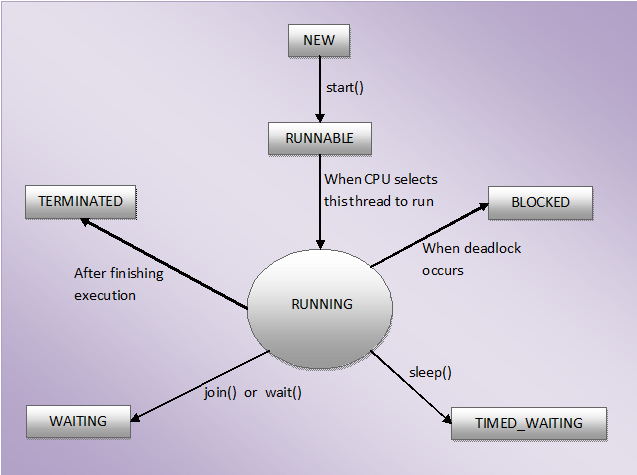
New: A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

Runnable: After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

Waiting: Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

Timed waiting: A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

Terminated ( Dead ): A runnable thread enters the terminated state when it completes its task or otherwise terminates.



Q: Method Overloading:

If a class has more than one method with same name, with different types of arguments, is called Method Overloading. Return type will not considered

Overloaded methods are differentiated by the number and the type of the arguments passed into the method. In the code sample, draw (String s) and draw (int i) are distinct and unique methods because they require different argument types.

You cannot declare more than one method with the same name and the same number and type of arguments, because the compiler cannot tell them apart.

The compiler does not consider return type when differentiating methods, so you cannot declare two methods with the same signature even if they have a different return type.

Q: Method Overriding.

If a class has same method name, and same number of and same type of Arguments of its super class is called method overriding.

An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

The ability of a subclass to override a method allows a class to inherit from a superclass whose behavior is "close enough" and then to modify behavior as needed.

Return type:

Parent return type: Number  
Child can be: Long/Integer/Double

If Parent return type: Long  
Child can be: Long

Q: What is a serialVersionUID and why should I use it?

The docs for [java.io.Serializable](http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html) are probably about as good an explanation as you'll get:

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then deserialization will result in anInvalidClassException. A serializable class can declare its own serialVersionUID explicitly by declaring a field named "serialVersionUID" that must be static, final, and of type long:

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

If a serializable class does not explicitly declare a serialVersionUID, then the serialization runtime will calculate a default serialVersionUID value for that class based on various aspects of the class, as described in the Java(TM) Object Serialization Specification. However, it is *strongly recommended* that all serializable classes explicitly declare serialVersionUID values, since 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. Therefore, to guarantee a consistent serialVersionUID value across different java compiler implementations, a serializable class must declare an explicit serialVersionUID value. It is also strongly advised that explicit serialVersionUID declarations use the private modifier where possible, since such declarations apply only to the immediately declaring class--serialVersionUID fields are not useful as inherited members.

Q: How does a Java HashMap handle different objects with the same hash code?

down voteaccepted

A hashmap works like this (this is a little bit simplified, but it illustrates the basic mechanism):

It has a number of "buckets" which it uses to store key-value pairs in. Each bucket has a unique number - that's what identifies the bucket. When you put a key-value pair into the map, the hashmap will look at the hash code of the key, and store the pair in the bucket of which the identifier is the hash code of the key. For example: The hash code of the key is 235 -> the pair is stored in bucket number 235. (Note that one bucket can store more then one key-value pair).

When you lookup a value in the hashmap, by giving it a key, it will first look at the hash code of the key that you gave. The hashmap will then look into the corresponding bucket, and then it will compare the key that you gave with the keys of all pairs in the bucket, by comparing them with equals().

Now you can see how this is very efficient for looking up key-value pairs in a map: by the hash code of the key the hashmap immediately knows in which bucket to look, so that it only has to test against what's in that bucket.

Looking at the above mechanism, you can also see what requirements are necessary on the hashCode() and equals() methods of keys:

* If two keys are the same (equals() returns true when you compare them), their hashCode()method must return the same number. If keys violate this, then keys that are equal might be stored in different buckets, and the hashmap would not be able to find key-value pairs (because it's going to look in the same bucket).
* If two keys are different, then it doesn't matter if their hash codes are the same or not. They will be stored in the same bucket if their hash codes are the same, and in this case, the hashmap will use equals() to tell them apart.

Q: Why does Java have transient fields?

The transient keyword in Java is used to indicate that a field should not be serialized.

Variables may be marked transient to indicate that they are not part of the persistent state of an object.

Q: Do you ever use the volatile keyword in Java?

volatile has semantics for memory visibility. Basically, the value of a volatile field becomes visible to all readers (other threads in particular) after a write operation completes on it. Without volatile, readers could see some non-updated value.

To answer your question: Yes, I use a volatile variable to control whether some code continues a loop. The loop tests the volatile value and continues if it is true. The condition can be set to false by calling a "stop" method. The loop sees false and terminates when it tests the value after the stop method completes execution.

The book "[Java Concurrency in Practice](http://jcip.net/)," which I highly recommend, gives a good explanation of volatile. This book is written by the same person who wrote the IBM article that is referenced in the question (in fact, he cites his book at the bottom of that article). My use of volatile is what his article calls the "pattern 1 status flag."

If you want to learn more about how [volatile](http://docs.oracle.com/javase/specs/jls/se8/html/jls-8.html#jls-8.3.1.4) works under the hood, read up on [the Java memory model](http://docs.oracle.com/javase/specs/jls/se8/html/jls-17.html). If you want to go beyond that level, check out a good computer architecture book like [Hennessy & Patterson](https://www.elsevier.com/books/computer-architecture/hennessy/978-0-12-383872-8) and read about cache coherence and cache consistency.

Q: Reverse a String in java?

**public** **class** ReverseString {

/\*\*

\* **@param** args

\*/

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

String string = "Dogs hates cats";

System.***out***.println("String Builder reverse..>" + **new** StringBuilder(string).reverse());

// char[] c=new char[string.length()];

**char**[] c = string.toCharArray();

**char** temp;

**for** (**int** i = 0; i < c.length / 2; i++) {

temp = c[i];

c[i] = c[c.length - 1 - i];

c[c.length - 1 - i] = temp;

}

System.***out***.println("Reverse String String...>" + **new** String(c));

}

}

Q: Encapsulation vs Abstraction?

Abstraction:

The process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface). The only good example i see for this across different sites is interface.

Encapsulation:

Its basically about hiding the state of object with the help of modifiers like private,public,protected etc. we expose the state thru public methods only if require.

**Encapsulation** is a strategy used as part of abstraction. Encapsulation refers to the state of objects - objects encapsulate their state and hide it from the outside; outside users of the class interact with it through its methods, but cannot access the classes state directly. So the class *abstracts* away the implementation details related to its state.

**Abstraction** is a more generic term, it can also be achieved by (amongst others) subclassing. For example, the interface List in the standard library is an abstraction for a sequence of items, indexed by their position, concrete examples of a List are an ArrayList or a LinkedList. Code that interacts with a List abstracts over the detail of which kind of a list it is using.

Abstraction is often not possible without hiding underlying state by encapsulation - if a class exposes its internal state, it can't change its inner workings, and thus cannot be abstracted.

**Abstraction** is the concept of describing something in simpler terms, i.e abstracting away the details, in order to focus on what is important (This is also seen in abstract art, for example, where the artist focuses on the building blocks of images, such as colour or shapes). The same idea translates to OOP by using an inheritance hierarchy, where more abstract concepts are at the top and more concrete ideas, at the bottom, build upon their abstractions. At its most abstract level there is no implementation details at all and perhaps very few commonalities, which are added as the abstraction decreases.

As an example, at the top might be an interface with a single method, then the next level, provides several abstract classes, which may or may not fill in some of the details about the top level, but branches by adding their own abstract methods, then for each of these abstract classes are concrete classes providing implementations of all the remaining methods.

**Encapsulation** is a *technique*. It may or may not be for aiding in abstraction, but it is certainly about information hiding and/or organisation. It demands data and functions be grouped in some way - of course good OOP practice demands that they should be grouped by abstraction. However, there are other uses which just aid in maintainability etc.

**Encapsulation** is a way to achieve [*"information hiding"*](http://en.wikipedia.org/wiki/Information_hiding) so, following your example, you don't *"need to know the internal working of the mobile phone to operate"* with it. You have an *interface* to use the device behaviour without knowing implementation details.

**Abstraction** on the other side, can be explained as the capability to use the same *interface* for different objects. Different implementations of the same interface can exist. Details are hidden by *encapsulation*.

**Abstraction** : you'll never buy a "device", but always buy something more specific : iPhone, GSII, Nokia 3310... Here, iPhone, GSII and N3310 are concrete things, device is abstract.

**Encapsulation** : you've got several devices, all of them have got an USB port. You don't know what kind of printed circuit there's back, you just have to know you'll be able to plug an USB cable onto.

Abstraction is a concept, which is allowed by encapsulation. My exemple wasn't the best one (there's no real link between the two blocks).

You can do encapsulation without using abstraction, but if you wanna use some abstraction in your projets, you'll need encapsulation.

Q: Abstract class vs Interface?

I will give you an example first:

public interface LoginAuth{

public String encryptPassword(String pass);

public void checkDBforUser();

}

Now suppose you have 3 databases in your application. Then each and every implementation for that database needs to define the above 2 methods:

public class DBMySQL implements LoginAuth{

// Needs to implement both methods

}

public class DBOracle implements LoginAuth{

// Needs to implement both methods

}

public class DBAbc implements LoginAuth{

// Needs to implement both methods

}

But what if encryptPassword() is not database dependent, and it's the same for each class? Then the above would not be a good approach.

Instead, consider this approach:

public abstract class LoginAuth{

public String encryptPassword(String pass){

// Implement the same default behavior here

// that is shared by all subclasses.

}

// Each subclass needs to provide their own implementation of this only:

public abstract void checkDBforUser();

}

Now in each child class, we only need to implement one method - the method that is database dependent.

I tried my best and Hope this will clear your doubts.

Q: Can An Interface have implementation?

Ans: till java 7, is not possible, from JDK 8, yes it is possible.

default and public static methods may have implementations  
and non-static public methods of interface do not have, implementation

Q: Abstract class vs Interface?

I was asked a question, I wanted to get my answer reviewed here.

**Q:** In which scenario it is more appropriate to extend an abstract class rather than implementing the interface(s)?

**A:** If we are using template method design pattern.

Am I correct ?

I am sorry if I was not able to state the question clearly.  
I know the basic difference between abstract class and interface.

1) use abstract class when the requirement is such that we need to implement the same functionality in every subclass for a specific operation (implement the method) and different functionality for some other operations (only method signatures)

2) use interface if you need to put the signature to be same (and implementation different) so that you can comply with interface implementation

3) we can extend max of one abstract class, but can implement more than one interface

Choosing between these two really depends on what you want to do, but luckily for us, Erich Gamma can help us a bit.

As always there is a trade-off, an interface gives you freedom with regard to the base class, an abstract class gives you the **freedom to add new methods later**. – Erich Gamma

You **can’t go and change an Interface without having to change a lot of other things** in your code, so the only way to avoid this would be to create a whole new Interface, which might not always be a good thing.

**When To Use Interfaces**

An interface allows somebody to start from scratch to implement your interface or implement your interface in some other code whose original or primary purpose was quite different from your interface. To them, your interface is only incidental, something that have to add on to the their code to be able to use your package. The disadvantage is every method in the interface must be public. You might not want to expose everything.

**When To Use Abstract classes**

An abstract class, in contrast, provides more structure. It usually defines some default implementations and provides some tools useful for a full implementation. The catch is, code using it must use your class as the base. That may be highly inconvenient if the other programmers wanting to use your package have already developed their own class hierarchy independently. In Java, a class can inherit from only one base class.

**When to Use Both**

You can offer the best of both worlds, an interface and an abstract class. Implementors can ignore your abstract class if they choose. The only drawback of doing that is calling methods via their interface name is slightly slower than calling them via their abstract class name.