**1.[What is a serialVersionUID and why should I use it?](http://stackoverflow.com/questions/285793/what-is-a-serialversionuid-and-why-should-i-use-it)**

**Ans:-**

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.

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 an InvalidClassException. 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.

Changes to a serializable class can be compatible or incompatible. Following is the list of changes which are compatible:

* Add fields
* Change a field from static to non-static
* Change a field from transient to non-transient
* Add classes to the object tree

List of incompatible changes:

* Delete fields
* Change class hierarchy
* Change non-static to static
* Change non-transient to transient
* Change type of a primitive field

2. **Write code to make an annotation ?**

Ans:-

This @interface tells Java this is a custom annotation. Later, you can annotate it on method level like this @Test(enable=false).

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD) *//can use in method only.*

**public** @**interface** Test {

*//should ignore this test?*

**public** **boolean** enabled() **default** **true**;

}

----------------------------------------------------------------------------

**This @TesterInfo is applied on class level, store the tester details. This shows the different use of return types – enum, array and string.**

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE) *//on class level*

**public** @**interface** TesterInfo {

**public** **enum** Priority {

LOW, MEDIUM, HIGH

}

Priority priority() **default** Priority.MEDIUM;

String[] tags() **default** "";

String createdBy() **default** "anil";

String lastModified() **default** "03/01/2014";

}

**Create a simple unit test example, and annotated with the new custom annotations – @Test and @TesterInfo.**

**import** com.mkyong.test.core.Test;

**import** com.mkyong.test.core.TesterInfo;

**import** com.mkyong.test.core.TesterInfo.Priority;

@TesterInfo(

priority = Priority.HIGH,

createdBy = "mkyong.com",

tags = {"sales","test" }

)

**public** **class** TestExample {

@Test

**void** testA() {

**if** (**true**)

**throw** **new** RuntimeException("This test always failed");

}

@Test(enabled = **false**)

**void** testB() {

**if** (**false**)

**throw** **new** RuntimeException("This test always passed");

}

@Test(enabled = **true**)

**void** testC() {

**if** (10 > 1) {

*// do nothing, this test always passed.*

}

}

}

**Below example show you how to use Java reflection APIs to read and process the custom annotations.**

**mport** java.lang.annotation.Annotation;

**import** java.lang.reflect.Method;

**import** com.mkyong.test.core.Test;

**import** com.mkyong.test.core.TesterInfo;

**public** **class** RunTest {

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

System.out.println("Testing...");

**int** passed = 0, failed = 0, count = 0, ignore = 0;

Class<TestExample> obj = TestExample.**class**;

*// Process @TesterInfo*

**if** (obj.isAnnotationPresent(TesterInfo.**class**)) {

Annotation annotation = obj.getAnnotation(TesterInfo.**class**);

TesterInfo testerInfo = (TesterInfo) annotation;

System.out.printf("%nPriority :%s", testerInfo.priority());

System.out.printf("%nCreatedBy :%s", testerInfo.createdBy());

System.out.printf("%nTags :");

**int** tagLength = testerInfo.tags().length;

**for** (String tag : testerInfo.tags()) {

**if** (tagLength > 1) {

System.out.print(tag + ", ");

} **else** {

System.out.print(tag);

}

tagLength--;

}

System.out.printf("%nLastModified :%s%n%n", testerInfo.lastModified());

}

*// Process @Test*

**for** (Method method : obj.getDeclaredMethods()) {

*// if method is annotated with @Test*

**if** (method.isAnnotationPresent(Test.**class**)) {

Annotation annotation = method.getAnnotation(Test.**class**);

Test test = (Test) annotation;

*// if enabled = true (default)*

**if** (test.enabled()) {

**try** {

method.invoke(obj.newInstance());

System.out.printf("%s - Test '%s' - passed %n", ++count, method.getName());

passed++;

} **catch** (Throwable ex) {

System.out.printf("%s - Test '%s' - failed: %s %n", ++count, method.getName(), ex.getCause());

failed++;

}

} **else** {

System.out.printf("%s - Test '%s' - ignored%n", ++count, method.getName());

ignore++;

}

}

}

System.out.printf("%nResult : Total : %d, Passed: %d, Failed %d, Ignore %d%n", count, passed, failed, ignore);

}

}

**Serialization**

**Q1) What is Serialization?**

Ans) Serializable is a marker interface. When an object has to be transferred over a network ( typically through rmi or EJB) or to persist the state of an object to a file, the object Class needs to implement Serializable interface. Implementing this interface will allow the object converted into bytestream and transfer over a network.

**Q2) What is use of serialVersionUID?**

Ans) During object serialization, the default Java serialization mechanism writes the metadata about the object, which includes the class name, field names and types, and superclass. This class definition is stored as a part of the serialized object. This stored metadata enables the deserialization process to reconstitute the objects and map the stream data into the class attributes with the appropriate type  
Everytime an object is serialized the java serialization mechanism automatically computes a hash value. ObjectStreamClass's *computeSerialVersionUID*() method passes the class name, sorted member names, modifiers, and interfaces to the secure hash algorithm (SHA), which returns a hash value.The serialVersionUID is also called *suid*.  
So when the serilaize object is retrieved , the JVM first evaluates the *suid* of the serialized class and compares the *suid* value with the one of the object. If the suid values match then the object is said to be compatible with the class and hence it is de-serialized. If not *InvalidClassException* exception is thrown.  
  
Changes to a serializable class can be compatible or incompatible. Following is the list of changes which are compatible:

* Add fields
* Change a field from static to non-static
* Change a field from transient to non-transient
* Add classes to the object tree

List of incompatible changes:

* Delete fields
* Change class hierarchy
* Change non-static to static
* Change non-transient to transient
* Change type of a primitive field

So, if no suid is present, inspite of making compatible changes, jvm generates new *suid* thus resulting in an exception if prior release version object is used .

The only way to get rid of the exception is to recompile and deploy the application again.

If we explicitly mention the sUid using the statement:

private final static long serialVersionUID = <integer value>

then if any of the metioned compatible changes are made the class need not to be recompiled. But for incompatible changes there is no other way than to compile again.

**Q3) What is the need of Serialization?**

Ans) The serialization is used :-

* To send state of one or more object’s state over the network through a socket.
* To save the state of an object in a file.
* An object’s state needs to be manipulated as a stream of bytes.

**4.Other than Serialization what are the different approach to make object Serializable?**

Ans:- Besides the Serializable interface, at least three alternate approaches can serialize Java objects:

* For object serialization, instead of implementing the Serializable interface, a developer can implement the Externalizable interface, which extends Serializable. By implementing Externalizable, a developer is responsible for implementing the writeExternal() and readExternal() methods. As a result, a developer has sole control over reading and writing the serialized objects.
* XML serialization is an often-used approach for data interchange. This approach lags runtime performance when compared with Java serialization, both in terms of the size of the object and the processing time. With a speedier XML parser, the performance gap with respect to the processing time narrows. Nonetheless, XML serialization provides a more malleable solution when faced with changes in the serializable object.
* Finally, consider a "roll-your-own" serialization approach. You can write an object's content directly via either the ObjectOutputStream or the DataOutputStream. While this approach is more involved in its initial implementation, it offers the greatest flexibility and extensibility. In addition, this approach provides a performance advantage over Java serialization.
* **Q5) Do we need to implement any method of Serializable interface to make an object serializable?**
* Ans) No. Serializable is a Marker Interface. It does not have any methods.
* **Q6) What happens if the object to be serialized includes the references to other serializable objects?**
* Ans) If the object to be serialized includes references to the other objects, then all those object’s state also will be saved as the part of the serialized state of the object in question. The whole object graph of the object to be serialized will be saved during serialization automatically provided all the objects included in the object’s graph are serializable.
* **Q7) What happens if an object is serializable but it includes a reference to a non-serializable object?**
* Ans- If you try to serialize an object of a class which implements serializable, but the object includes a reference to an non-serializable class then a ‘NotSerializableException’ will be thrown at runtime.
* public class NonSerial {
* //This is a non-serializable class
* }
* public class MyClass implements Serializable {
* private static final long serialVersionUID = 1L;
* private NonSerial nonSerial;
* MyClass(NonSerial nonSerial){
* this.nonSerial = nonSerial;
* }
* public static void main(String [] args) {
* NonSerial nonSer = new NonSerial();
* MyClass c = new MyClass(nonSer);
* try {
* FileOutputStream fs = new FileOutputStream("test1.ser");
* ObjectOutputStream os = new ObjectOutputStream(fs);
* os.writeObject(c);
* os.close();
* } catch (Exception e) { e.printStackTrace(); }
* try {
* FileInputStream fis = new FileInputStream("test1.ser");
* ObjectInputStream ois = new ObjectInputStream(fis);
* c = (MyClass) ois.readObject();
* ois.close();
* } catch (Exception e) {
* e.printStackTrace();
* }
* }
* }
* On execution of above code following exception will be thrown;
* java.io.NotSerializableException: NonSerial
* at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java)
* **Q8) Are the static variables saved as the part of serialization?**
* Ans) No. The static variables belong to the class are not the part of the state of the object so they are not saved as the part of serialized object.
* **Q9)What is a transient variable?**
* Ans) These variables are not included in the process of serialization and are not the part of the object’s serialized state.
* **Q10) What will be the value of transient variable after de-serialization?**
* Ans) It’s default value.  
  e.g. if the transient variable in question is an int, it’s value after deserialization will be zero.
* public class TestTransientVal implements Serializable {
* private static final long serialVersionUID = -22L;
* private String name;
* transient private int age;
* TestTransientVal(int age, String name) {
* this.age = age;
* this.name = name;
* }
* public static void main(String [] args) {
* TestTransientVal c = new TestTransientVal(1,"ONE");
* System.out.println("Before serialization:" + c.name + " "+ c.age);
* try {
* FileOutputStream fs =new FileOutputStream("testTransient.ser");
* ObjectOutputStream os = new ObjectOutputStream(fs);
* os.writeObject(c);
* os.close();
* } catch (Exception e) { e.printStackTrace(); }
* try {
* FileInputStream fis =new FileInputStream("testTransient.ser");
* ObjectInputStream ois =new ObjectInputStream(fis);
* c = (TestTransientVal) ois.readObject();
* ois.close();
* } catch (Exception e) { e.printStackTrace(); }
* System.out.println("After de-serialization:" + c.name +" "+ c.age);
* }
* }
* Result of executing above piece of code –  
  Before serialization: - Value of non-transient variable ONE Value of transient variable 1   
  After de-serialization:- Value of non-transient variable ONE Value of transient variable 0
* Explanation –   
  The transient variable is not saved as the part of the state of the serailized variable, it’s value after de-serialization is it’s default value.
* **Q11) Does the order in which the value of the transient variables and the state of the object using the defaultWriteObject() method are saved during serialization matter?**
* Ans) Yes, while restoring the object’s state the transient variables and the serializable variables that are stored must be restored in the same order in which they were saved.
* **Q12) How can one customize the Serialization process? or What is the purpose of implementing the writeObject() and readObject() method?**
* Ans) When you want to store the transient variables state as a part of the serialized object at the time of serialization the class must implement the following methods –
* private void wrtiteObject(ObjectOutputStream outStream) {
* //code to save the transient variables state
* //as a part of serialized object
* }
* private void readObject(ObjectInputStream inStream) {
* //code to read the transient variables state
* //and assign it to the de-serialized object
* }
* public class TestCustomizedSerialization implements Serializable {
* private static final long serialVersionUID =-22L;
* private String noOfSerVar;
* transient private int noOfTranVar;
* TestCustomizedSerialization(int noOfTranVar, String noOfSerVar) {
* this.noOfTranVar = noOfTranVar;
* this.noOfSerVar = noOfSerVar;
* }
* private void writeObject(ObjectOutputStream os) {
* try {
* os.defaultWriteObject();
* os.writeInt(noOfTranVar);
* } catch (Exception e) { e.printStackTrace(); }
* }
* private void readObject(ObjectInputStream is) {
* try {
* is.defaultReadObject();
* int noOfTransients = (is.readInt());
* } catch (Exception e) {
* e.printStackTrace(); }
* }
* public int getNoOfTranVar() {
* return noOfTranVar;
* }
* The value of transient variable ‘noOfTranVar’ is saved as part of the serialized object manually by implementing writeObject() and restored by implementing readObject().  
  The normal serializable variables are saved and restored by calling defaultWriteObject() and defaultReadObject()respectively. These methods perform the normal serialization and de-sirialization process for the object to be saved or restored respectively.
* **Q13) If a class is serializable but its superclass in not, what will be the state of the instance variables inherited from super class after deserialization?**
* Ans) The values of the instance variables inherited from superclass will be reset to the values they were given during the original construction of the object as the non-serializable super-class constructor will run.
* E.g.
* public class ChildSerializable extends ParentNonSerializable implements Serializable {
* private static final long serialVersionUID = 1L;
* String color;
* ChildSerializable() {
* this.noOfWheels = 8;
* this.color = "blue";
* }
* }
* public class SubSerialSuperNotSerial {
* public static void main(String [] args) {
* ChildSerializable c = new ChildSerializable();
* System.out.println("Before : - " + c.noOfWheels + " "+ c.color);
* try {
* FileOutputStream fs = new FileOutputStream("superNotSerail.ser");
* ObjectOutputStream os = new ObjectOutputStream(fs);
* os.writeObject(c);
* os.close();
* } catch (Exception e) { e.printStackTrace(); }
* try {
* FileInputStream fis = new FileInputStream("superNotSerail.ser");
* ObjectInputStream ois = new ObjectInputStream(fis);
* c = (ChildSerializable) ois.readObject();
* ois.close();
* } catch (Exception e) { e.printStackTrace(); }
* System.out.println("After :- " + c.noOfWheels + " "+ c.color);
* }
* }
* Result on executing above code –
* Before : - 8 blue
* After :- 4 blue
* The instance variable ‘noOfWheels’ is inherited from superclass which is not serializable. Therefore while restoring it the non-serializable superclass constructor runs and its value is set to 8 and is not same as the value saved during serialization which is 4.
* **Q14) To serialize an array or a collection all the members of it must be serializable. True /False?**
* Ans) True.

**Keywords combination for serialization**

//not participates in serialization

//private static int pinCode =10032;

//participates in serialization

**private** **final** **static** **int** *pinCode* =10032;

//participates in serialization

//private final int pinCode =10032;

//Participates in serialization

**private** **transient** **static** **int** *age*;

// not participate in Serialization

// private transient final int age;

**Collections**

**HashMap**

# [How HashMap works in Java](http://javarevisited.blogspot.in/2011/02/how-hashmap-works-in-java.html)

**How HashMap works in Java or sometime how get method work in HashMap is a very common question on Java interviews now days. Almost everybody who worked in Java knows about HashMap, where to use HashMap and difference between Hashtable and HashMap then why this interview question becomes so special? Because of the depth it offers. It has become very popular Java interview question in almost any senior or mid-senior level Java interviews. Investment banks mostly prefer to ask this question and some time even ask you to implement your own HashMap based upon your coding aptitude. Introduction of [ConcurrentHashMap](http://javarevisited.blogspot.co.uk/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) and other concurrent collections has also made this questions as starting point to delve into more advanced feature. let's start the journey.**

## How HashMap Internally Works in Java

**Questions start with simple statement :**

**Have you used HashMap before or  What is HashMap? Why do you use it**

**Almost everybody answers this with yes and then interviewee keep talking about common facts about HashMap like HashMap accept null while Hashtable doesn't, [HashMap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), HashMap is fast and so on along with basics like its stores key and value pairs etc. This shows that person has used HashMap and quite familiar with the functionality it offers, but interview takes a sharp turn from here and next set of follow-up questions gets more detailed about fundamentals involved with HashMap in Java . Interviewer strike back with questions like :**

**Do you Know how HashMap works in Java or How does get () method of HashMap works in Java**

**And then you get answers like,  I don't bother its standard Java API, you better look code on Java source or Open JDK; I can find it out in Google at any time etc. But some interviewee definitely answer this and will say HashMap works on principle of hashing, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object  to put() method on Java HashMap, HashMap implementation calls [hashCode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in bucket which is essential to understand the retrieving logic. If people fails to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in Java HashMap . This answer is very much acceptable and does make sense that interviewee has fair bit of knowledge on how hashing works and how HashMap  works in Java. But this is just start of story and confusion increases when you put interviewee on scenarios faced by Java developers on day by day basis. Next question could be about collision detection and collision resolution in Java HashMap  e.g.**

**What will happen if two different objects have same hashcode?**

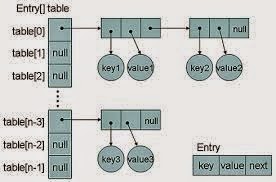
**Now from here onwards real confusion starts, Some time candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about** [**equals() and hashCode() contract**](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) **that two unequal object in Java can have same hash code. Some will give up at this point and few will move ahead and say "Since hashcode is same, bucket location would be same and collision will occur in HashMap, Since HashMap use LinkedList to store object, this entry (object of Map.Entry comprise key and value )  will be stored in** [**LinkedList**](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html)**. Great this answer make sense though there are many collision resolution methods available  like linear probing and chaining, this is simplest and HashMap in Java does follow this. But story does not end here and interviewer asks**

**How will you retrieve Value object  if two Keys will have same hashcode?**

**[how HashMap works internally in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)Interviewee will say we will call get() method and then HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but then you need to remind him that there are two Value objects are stored in same bucket , so they will say about [traversal in LinkedList](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html) until we find the value object , then you ask *how do you identify value object because you don't  have value object to compare* ,Until they know that HashMap  stores both Key and Value in LinkedList node or as Map.Entry they won't be able to resolve this issue and will try and fail.**

**But those bunch of people who remember this key information will say that after finding bucket location , we will call keys.equals() method to identify correct node in LinkedList and return associated value object for that key in Java HashMap . Perfect this is the correct answer.**

**In many cases interviewee fails at this stage because they get confused between [hashCode()](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) and equals() or keys and values object in Java HashMap  which is pretty obvious because they are dealing with the hashcode() in all previous questions and equals() come in picture only in case of retrieving value object from HashMap in Java. Some good developer point out here that using immutable, [final object](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) with proper equals() and hashcode() implementation would act as perfect Java HashMap  keys and improve performance of Java HashMap  by reducing collision. Immutability *also allows caching there hashcode of different keys* which makes overall retrieval process very fast and suggest that [String](http://javarevisited.blogspot.sg/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) and various wrapper classes e.g. Integer very good keys in Java HashMap.**

**[](http://4.bp.blogspot.com/-adRczhctozE/VD_eimhTQbI/AAAAAAAACCg/lfA1G5GZXyM/s1600/How%2BHashMap%2Bworks%2Bin%2BJava%2B%281%29.jpg)**

**Now if you clear this entire Java HashMap interview,  You will be surprised by this very interesting question "What happens On HashMap in Java if the size of the HashMap  exceeds a given threshold defined by load factor ?". Until you know how HashMap  works exactly you won't be able to answer this question. If the size of the Map exceeds a given threshold defined by load-factor e.g. if load factor is .75 it will act to re-size the map once it filled 75%. Similar to other collection classes like** [**ArrayList**](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html)**,  Java HashMap re-size itself by creating a new bucket array of size twice of previous size of HashMap , and then start putting every old element into that new bucket array. This process is called rehashing because it also applies hash function to find new bucket location.**

**If you manage to answer this question on HashMap in Java you will be greeted by "do you see any problem with resizing of HashMap  in Java" , you might not be able to pick the context and then he will try to give you hint about multiple thread accessing the Java HashMap and potentially looking for race condition on HashMap  in Java.**

**So the answer is Yes there is potential** [**race condition**](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) **exists while resizing HashMap in Java, if two** [**thread**](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) **at the same time found that now HashMap needs resizing and they both try to resizing. on the process of resizing of HashMap in Java , the element in bucket which is stored in linked list get reversed in order during there migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at head *to avoid tail traversing*. If race condition happens then you will end up with an infinite loop. Though this point you can potentially argue that what the hell makes you think to use HashMap  in multi-threaded environment to interviewer :)**

## Some more Hashtable and HashMap Questions

**Few more question on HashMap in Java which is contributed by readers of Javarevisited blog :**

**1) Why String, Integer and other wrapper classes are considered good keys ?**

**String, Integer and other wrapper classes are natural candidates of HashMap key, and String is most frequently used key as well because** [**String is immutable and final**](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html)**,and overrides equals and hashcode() method. Other wrapper class also shares similar property. Immutabiility is required, in order to prevent changes on fields used to calculate hashCode() because if key object return different hashCode during insertion and retrieval than it won't be possible to get object from HashMap. Immutability is best as it offers other advantages as well like** [**thread-safety**](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html)**, If you can keep your hashCode same by only making certain fields final, then you go for that as well. Since equals() and hashCode() method is used during reterival of value object from HashMap, its important that key object correctly override these methods and follow contact. If unequal object return different hashcode than chances of collision will be less which subsequently improve performance of HashMap.**

**2) Can we use any custom object as key in HashMap ?**

**This is an extension of previous questions. Ofcourse you can use any Object as key in Java HashMap provided it follows equals and hashCode contract and its hashCode should not vary once the object is inserted into** [**Map**](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)**. If custom object is Immutable than this will be already taken care because you can not change it once created.**

**3) Can we use ConcurrentHashMap in place of Hashtable ?**

**This is another question which getting popular due to increasing popularity of ConcurrentHashMap. Since we know Hashtable is synchronized but ConcurrentHashMap provides better concurrency by only locking portion of map determined by concurrency level. ConcurrentHashMap is certainly introduced as Hashtable and can be used in place of it but Hashtable provide stronger thread-safety than ConcurrentHashMap. See my post** [**difference between Hashtable and ConcurrentHashMap**](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) **for more details.**

**Personally, I like this question because of its depth and number of concept it touches indirectly, if you look at questions asked during interview this HashMap  questions has verified**

* **Concept of hashing**
* **Collision resolution in HashMap**
* **Use of equals () and hashCode () and there importance in HashMap?**
* **Benefit of immutable object?**
* **Race condition on HashMap  in Java**
* **Resizing of Java HashMap**

**Just to summarize here are the answers which does makes sense for above questions**

**How HashMap  works in Java**

**HashMap  works on principle of hashing, we have put() and get() method for storing and retrieving object form HashMap .When we pass an both key and value to put() method to store on HashMap , it uses key object hashcode() method to calculate hashcode and they by applying hashing on that hashcode it identifies bucket location for storing value object. While retrieving it uses key object equals method to find out correct key value pair and return value object associated with that key. HashMap  uses linked list in case of collision and object will be stored in next node of linked list. Also [HashMap stores both key and value tuple](http://java67.blogspot.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) in every node of linked list in form of Map.Entry object.**

**What will happen if two different HashMap  key objects have same hashcode?**

**They will be stored in same bucket but no next node of linked list. And keys equals () method will be used to identify correct key value pair in HashMap .**

**In terms of usage Java HashMap is very versatile and I have mostly used HashMap as cache in electronic trading application I have worked . Since finance domain used Java heavily and due to performance reason we need caching HashMap and ConcurrentHashMap  comes as very handy there. You can also check following articles form Javarevisited to learn more about HashMap and Hashtable in Java :**

## HashMap Changes in JDK 1.7 and JDK 1.8

**There is some [performance improvement done on HashMap and ArrayList from JDK 1.7](http://javarevisited.blogspot.com/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html), which reduce memory consumption. Due to this empty Map are lazily initialized and will cost you less memory. Earlier, when you create HashMap e.g. new HashMap() it automatically creates array of default length e.g. 16. After some research, Java team founds that most of this Map are temporary and never use that many elements, and only end up wasting memory. Also, From JDK 1.8 onwards HashMap has introduced an improved strategy to deal with high collision rate. Since a poor hash function e.g. which always return location of same bucket, can turn a HashMap into linked list, i.e. converting get() method to perform in O(n) instead of O(1) and someone can take advantage of this fact, Java now internally replace linked list to a binary true once certain threshold is breached. This ensures performance or order O(log(n)) even in worst case where hash function is not distributing keys properly.**

**Interview questions from Collection** package or framework is most common in any Core Java Interview yet a tricky one. Together Collection and multithreading makes any Java interview tough to crack and having a good understanding of Collection and threads will help you to excel in Java interview. I thought about writing interview questions on collection when I wrote [10 multi-threading Interview questions](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html) and [Top 20 Core Java Interview questions answers](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) but somehow it got delayed. In this article we will see mix of some beginners and advanced Java Collection interviews and there answers which has been asked in various Core Java interviews. These Collection interview questions have been collected from various friends and colleagues and Answers of these interview questions can also be found by Google.

## Good Java Collection Interview Questions Answers

[Java Collection interview questions answers](http://javarevisited.blogspot.com/2011/10/jsp-interview-questions-answers-for.html)Now let's start with *interview questions on collections*. Since collection is made of various data structures e.g. Map, Set and List and there various implementation, mostly interviewer checks whether interviewee is familiar with basics of these collections or not and whether he knows when to use Map, Set or List. Based on Role for which interview is going on questions starts with beginner’s level or more advanced level. Normally 2 to 3 years experience counted as beginners while over 5 years comes under advanced category, we will see questions from both categories.

**1. How HashMap works in Java?**

This is *Classical Java Collection interview questions* which I have also discussed in [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html). This collection interview questions is mostly asked during AVP Role interviews on Investment-Banks and has lot of follow-up questions based on response of interviewee e.g. Why HashMap keys needs to be [immutable](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html), what is race conditions on HashMap and how HashMap resize in Java. For explanation and answers of these questions Please see earlier link.

**2. What is difference between poll() and remove() method of Queue interface?**  
Though both poll() and remove() method from Queue is used to remove object and returns head of the queue, there is subtle difference between them. If Queue is empty() then a call to remove() method will throw Exception, while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon queue's ordering policy and varies between different implementation, for example PriorityQueue keeps lowest element as per Comparator or Comparable at head position. 

**3. What is difference between fail-fast and fail-safe Iterators?**

This is relatively *new collection interview questions* and can become trick if you hear the term fail-fast and fail-safe first time. Fail-fast Iterators throws ConcurrentModificationException when one [Thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) is iterating over collection object and other thread structurally modify Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure. On the other hand [fail-safe Iterators](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) works on copy of collection instead of original collection

**4. How do you remove an entry from a Collection? and subsequently what is difference between remove() method of Collection and remove() method of Iterator, which one you will use, while removing elements during iteration?**  
  
Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these method to remove an entry from Collection, while not iterating. Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

**5. What is difference between Synchronized Collection and Concurrent Collection?**

Java 5 has added several new Concurrent Collection classes e.g. ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc, which has made Interview questions on Java Collection even trickier. Java Also provided way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.synchronizedMap() Utility function.One Significant difference is that Concurrent Collections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization. See [Difference between Synchronized Collection and Concurrent Collection in Java](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) for more details.

**6. What is difference between Iterator and Enumeration?**

This is a beginner level collection interview questions and mostly asked during interviews of Junior Java developer up to experience of 2 to 3 years Iterator duplicate functionality of Enumeration with one addition of remove() method and both provide navigation functionally on objects of Collection.Another difference is that Iterator is more safe than Enumeration and doesn't allow another thread to modify collection object during iteration except remove() method and throws ConcurrentModificaitonException. See [Iterator vs Enumeration in Java](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html) for more differences.

**7. How does HashSet is implemented in Java, How does it uses Hashing ?**  
This is a tricky question in Java, because for hashing you need both key and value and there is no key for store it in a bucket, then how exactly HashSet store element internally. Well, HashSet is built on top of HashMap. If you look at source code of java.util.HashSet class, you will find that that it uses a HashMap with same values for all keys, as shown below :  
  
private transient HashMap map;  
  
// Dummy value to associate with an Object in the backing Map  
private static final Object PRESENT = new Object();  
  
When you call add() method of HashSet, it put entry in HashMap :  
  
public boolean add(E e) {  
  return map.put(e, PRESENT)==null;  
}  
  
Since keys are unique in a HashMap, it provides uniqueness guarantee of Set interface.  
  
  
**8. What do you need to do to use a custom object as key in Collection classes like Map or Set?**  
Answer is : If you are using any custom object in Map as key, you need to override equals() and hashCode() method, and make sure they follow there contract. On the other hand if you are storing a custom object in Sorted Collection e.g. SortedSet or SortedMap, you also need to make sure that your equals() method is consistent to compareTo() method, otherwise those collection will not follow there contacts e.g. Set may allow duplicates.

**9. Difference between HashMap and Hashtable?**

This is another Classical Java Collection interview asked on beginner’s level and most of Java developer has a predefined answer for this interview questions e.g. HashMap is not synchronized while Hashtable is not or hashmap is faster than hash table etc. What could go wrong is that if he placed another follow-up question like how hashMap works in Java or can you replace Hashtable with ConcurrentHashMap etc. See [Hashtable vs HashMap in Java](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) for detailed answer of this interview question.

**10. When do you use ConcurrentHashMap in Java?**

This is another advanced level collection interview questions in Java which normally asked to check whether interviewer is familiar with optimization done on ConcurrentHashMap or not. ConcurrentHashMap is better suited for situation where you have multiple readers and one

Writer or fewer writers since Map gets locked only during write operation. If you have equal number of reader and writer than [ConcurrentHashMap](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) will perform in line of Hashtable or synchronized HashMap.

**11. What is difference between Set and List in Java?**

Another classical Java Collection interview popular on telephonic round or first round of interview. Most of Java programmer knows that Set doesn't allowed duplicate while List does and List maintains insertion order while Set doesn't. What is key here is to show interviewer that you can decide which collection is more suited based on requirements.

**12. How do you Sort objects on collection?**

This Collection interview question serves two purpose it not only test an important programming concept Sorting but also utility class like Collections which provide several methods for creating synchronized collection and sorting. Sorting is implemented using Comparable and Comparator in Java and when you call Collections.sort() it gets sorted based on natural order specified in compareTo() method while Collections.sort(Comparator) will sort objects based on compare() method of Comparator. See [Sorting in Java using Comparator and Comparable](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) for more details.

**13. What is difference between Vector and ArrayList?**

One more beginner level collection interview questions, this is still very popular and mostly asked in telephonic round. [ArrayList in Java](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) is one of the most used Collection class and most interviewer asked questions on ArrayList. See Difference between Vector and ArrayList for answer of this interview question.

**14. What is difference between HashMap and HashSet?**

This collection interview questions is asked in conjunction with HashMap vs Hashtable. HashSet implements java.util.Set interface and that's why only contains unique elements, while HashMap allows duplicate values.  In fact, HashSet is actually implemented on top of java.util.HashMap. If you look internal implementation of java.util.HashSet, you will find that it adds element as key on internal map with same values. For a more detailed answer, see [HashMap vs HashSet](http://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html).  
  
  
  
**15) What is NavigableMap in Java ? What is benefit over Map?**  
NavigableMap Map was added in Java 1.6, it adds navigation capability to Map data structure. It provides methods like lowerKey() to get keys which is less than specified key, floorKey() to return keys which is less than or equal to specified key, ceilingKey() to get keys which is greater than or equal to specified key and higherKey() to return keys which is greater specified key from a Map. It also provide similar methods to get entries e.g. lowerEntry(), floorEntry(), ceilingEntry() and higherEntry(). Apart from navigation methods, it also provides utilities to create sub-Map e.g. creating a Map from entries of an exsiting Map like tailMap, headMap and subMap. headMap() method returns a NavigableMap whose keys are less than specified, tailMap() returns a NavigableMap whose keys are greater than the specified and subMap() gives a NavigableMap between a range, specified by toKey to fromKey.    
  
  
  
**16) Which one you will prefer between Array and ArrayList for Storing object and why?**Though ArrayList is also backed up by array, it offers some usability advantage over array in Java. Array is fixed length data structure, once created you can not change it's length. On the other hand, ArrayList is dynamic, it automatically allocate a new array and copies content of old array, when it resize. Another reason of using ArrayList over Array is support of Generics. Array doesn't support Generics, and if you store an Integer object on a String array, you will only going to know about it at runtime, when it throws ArrayStoreException. On the other hand, if you use ArrayList, compiler and IDE will catch those error on the spot. So if you know size in advance and you don't need re-sizing than use array, otherwise use ArrayList.

**17) Can we replace Hashtable with ConcurrentHashMap?**

Answer 3 : Yes we can replace Hashtable with ConcurrentHashMap and that's what suggested in Java documentation of ConcurrentHashMap. but you need to be careful with code which relies on locking behavior of Hashtable. Since Hashtable locks whole Map instead of portion of Map, compound operations like if(Hashtable.get(key) == null) put(key, value) works in Hashtable but not in concurrentHashMap. instead of this use putIfAbsent() method of ConcurrentHashMap

**18) What is CopyOnWriteArrayList, how it is different than ArrayList and Vector?**

Answer : CopyOnWriteArrayList is new List implementation introduced in Java 1.5 which provides better concurrent access than Synchronized List. better concurrency is achieved by Copying ArrayList over each write and replace with original instead of locking. Also CopyOnWriteArrayList doesn't throw any ConcurrentModification Exception. Its different than ArrayList because its thread-safe and ArrayList is not thread safe and its different than Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers.

**19) Why ListIterator has add() method but Iterator doesn't or Why add() method is declared in ListIterator and not on Iterator.**

Answer : ListIterator has add() method because of its ability to traverse or iterate in both direction of collection. it maintains two pointers in terms of previous and next call and in position to add new element without affecting current iteration.

**20) When does ConcurrentModificationException occur on iteration?**

When you remove object using Collection's or List's remove method e.g. remove(Object element) or remove(int index), instead of Iterator's remove() method than ConcurrentModificationException occur. As per Iterator's contract, if it detect any structural change in Collection e.g. adding or removing of element, once Iterator begins, it can throw ConcurrentModificationException.

**21) Difference between Set, List and Map Collection classes?**

java.util.Set, java.util.List and java.util.Map defines three of most popular data structure support in Java. Set provides uniqueness guarantee i.e.g you can not store duplicate elements on it, but it's not ordered. On the other hand List is an ordered Collection and also allowes duplicates. Map is based on hashing and stores key and value in an Object called entry. It provides O(1) performance to get object, if you know keys, if there is no collision. Popular impelmentation of Set is HashSet, of List is ArrayList and LinkedList, and of Map are HashMap, Hashtable and ConcurrentHashMap. Another key difference between Set, List and Map are that Map doesn't implement Collection interface, while other two does. For a more detailed answer, see Set vs List vs Map in Java

**22) What is BlockingQueue, how it is different than other collection classes?**

BlockingQueue is a Queue implementation available in java.util.concurrent package. It's one of the concurrent Collection class added on Java 1.5, main difference between BlockingQueue and other collection classes is that apart from storage, it also provides flow control. It can be used in inter thread communication and also provides built-in thread-safety by using happens-before guarantee. You can use BlockingQueue to solve Producer Consumer problem, which is what is needed in most of concurrent applications.

Read more: <http://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html#ixzz3IlOIyHzu>

**String**

[**Why String is immutable or final in Java**](http://javarevisited.blogspot.in/2010/10/why-string-is-immutable-in-java.html)

This is one of the most popular [String Interview questions in Java](http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html), which starts with discussion of,  What is String, How String in Java is different than String in C and C++, and then shifted towards [what is immutable object in Java](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html) , what are the benefits of immutable object , why do you use it and which scenarios do you use it. This is some time also asked as *"Why String is final in Java"* . Though there could be many possible answer for this question, and only designer of String class can answer this , I think below two does make sense  
  
1) Imagine StringPool facility without making string immutable , its not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many [reference variables](http://javarevisited.blogspot.sg/2012/02/difference-between-instance-class-and.html) , so if any one of them change the value others will be automatically gets affected i.e. lets say  
  
String A = "Test"  
String B = "Test"   
  
Now String B called "Test".toUpperCase() which change the same object into "TEST" , so A will also be "TEST" which is not desirable.  
  
2)String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string , you can pass database URL as string for opening database connection, you can [open any file in Java](http://javarevisited.blogspot.sg/2012/07/read-file-line-by-line-java-example-scanner.html) by passing name of file as argument to File I/O classes.  
  
In case, if String is not immutable, this would lead serious security threat , I mean some one can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access of those file. Because of immutability, you don't need to worry about those kind of threats. This reason also gel with, **Why String is final in Java**, by making java.lang.String final, Java designer ensured that no one overrides any behavior of String class.  
  
3)Since String is immutable it can safely shared between many threads ,which is very important for multithreaded programming and to avoid any [synchronization issues in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), Immutability also makes String instance [thread-safe in Java](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), means you don't need to synchronize String operation externally. Another important point to note about String is [memory leak caused by SubString](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html), which is not a thread related issues but something to be aware of.  
  
4) Another reason of **Why String is immutable in Java** is to **allow String to cache its hashcode** , being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in [hashmap in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html).  This one is also suggested by  Jaroslav Sedlacek in comments below. In short because String is immutable, no one can change its contents once created which guarantees [hashCode](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) of String to be same on multiple invocation.  
  
5) Another good reason of Why String is immutable in Java suggested by Dan Bergh Johnsson on comments is: The absolutely most important reason that String is immutable is that it is used by the [class loading mechanism](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter"

Read more: [http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html#ixzz3IlNMBBJG](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html" \l "ixzz3IlNMBBJG)

**MULTITHREADING**

# [How to Use Locks in Multi-threaded Java Program](http://javarevisited.blogspot.in/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html)

Many Java programmers confused themselves like hell while writing multi-threaded Java programs e.g. where to synchronized? Which Lock to use? What Lock to use etc. I often receive request to explain about *how to use Locks in Java*, so I thought to write a simple Java program, which is multi-threaded and uses rather new Lock interface. Remember Lock is your tool to guard shared resource which can be anything e.g. database, File system, a Prime number Generator or a Message processor. Before using Locks in Java program, it’s also better to learn some basics. Lock is an interface from java.util.concurrent package. It was introduced in JDK 1.5 release as an alternative of synchronized keyword. If you have never written any multi-threading program, then I suggest first start with synchronized keyword because it’s easier to use them. Once you are familiar with working of multi-threading program e.g. How threads share data, [how inter thread communication works](http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html), you can start with Lock facility. As I told you Lock is an interface, so we cannot use it directly, instead we need to use its implementation class. Thankfully Java comes with two implementation of java.util.concurrent.locks.Lock interface, ReentrantLock and ReentrantReadWriteLock, later provides two more inner implementation known as ReentrantReadWriteLock.ReadLock and ReentrantReadWriteLock.WriteLock. For our simple multi-threaded Java program's purpose ReentrantLock is enough.  
Here is the idiom to use Locks in Java :

Lock l = ...;

l.lock();

**try** {

// access the resource protected by this lock

} **finally** {

l.unlock();

}

You can see that Lock is used to protect a resource, so that only one thread can access it at a time. Why we do that? to make sure our application behave properly. For example we can use Lock to protect a counter, whose sole purpose is to return a count incremented by one, when anyone calls its getCount() method. If we don't protect them by [parallel access of thread](http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html), then it’s possible that two thread receives same count, which is against the program's policies. Now, coming back to semantics, we have used lock() method to acquire lock and unlock() method to release lock. Always remember to release lock in finally block, because every object has only one lock and if a thread doesn't release it then no one can get it, which may result in your program hung or threads going into deadlock. That's why I said that synchronized keyword is simpler than lock, because Java itself make sure that lock acquired by thread by entering into synchronized block or method is released as soon as it came out of the block or method. This happens even if thread came out by throwing exception, this is also we have [unlock code in finally block](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html), to make sure it run even if try block throws exception or not. In next section we will see example of our multi-threaded Java program, which uses Lock to protect shared Counter.

## Java Lock and ReentrantLock Example

Here is a sample Java program, which uses both Lock and ReentrantLock to protect a shared resource. In our case it’s an object, a counter's object. Invariant of Counter class is to return a count incremented by 1 each time someone calls getCount() method. Here for testing three threads will call getCount() method simultaneously but guard provided by Lock will prevent shared counter. As an exercise you can also implement same class using [synchronized keyword](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html). Here is complete code :

**import** **java.util.concurrent.locks.Lock**;

**import** **java.util.concurrent.locks.ReentrantLock**;

/\*\*

\*

\* Java Program to show how to use Locks in multi-threading

\* e.g. ReentrantLock, ReentrantReadWriteLock etc.

\*

\* @author Javin Paul

\*/

**public** **class** **LockDemo** {

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

// Let's create a counter and shared it between three threads

// Since Counter needs a lock to protect its getCount() method

// we are giving it a ReentrantLock.

**final** Counter myCounter = **new** Counter(**new** ReentrantLock());

// Task to be executed by each thread

Runnable r = **new** Runnable() {

**@Override**

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

System.out.printf("Count at thread %s is %d %n",

Thread.currentThread().getName(), myCounter.getCount());

}

};

// Creating three threads

Thread t1 = **new** Thread(r, "T1");

Thread t2 = **new** Thread(r, "T2");

Thread t3 = **new** Thread(r, "T3");

//starting all threads

t1.start();

t2.start();

t3.start();

}

}

**class** **Counter** {

**private** Lock lock; // Lock to protect our counter

**private** **int** count; // Integer to hold count

**public** **Counter**(Lock myLock) {

**this**.lock = myLock;

}

**public** **final** **int** **getCount**() {

lock.lock();

**try** {

count++;

**return** count;

} **finally** {

lock.unlock();

}

}

}

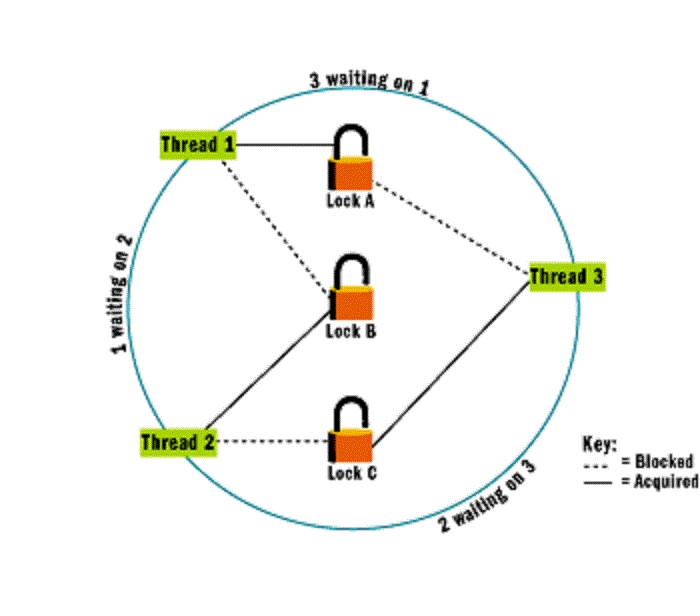
**Output:**

Count at thread T1 is **1**

Count at thread T2 is **2**

Count at thread T3 is **3**

You can even put a long loop inside Runnable's run() method to call getCount() numerous time, if you see a duplicate means there is a problem with your code, but without any duplicate means it’s working fine.

[](http://1.bp.blogspot.com/-7MKR8YVaC9c/VD0v4-AFOII/AAAAAAAACCQ/MkIqUKhlf2g/s1600/Lock%2BExample%2Bin%2BJava.gif)

## Common Mistakes made by beginners while using Locks in Java

Here are some of the common mistakes I have observed by looking at Java beginners lock related code :  
  
1) Instead of sharing lock they provide different locks to each thread. This often happens to them unknowingly because they usually put the lock and guarded block inside Runnable, and they pass two [different instances of Runnable](http://java67.blogspot.com/2012/08/what-is-thread-and-runnable-in-java.html) to two different threads e.g. where SimpleLock is a Runnable, as shown below :

Thread firstThread = **new** Thread(**new** SimpleLock());

Thread secondThread = **new** Thread(**new** SimpleLock());

**class** **SimpleLock** **implements** Runnable {

**private** Lock myLock = **new** ReentrantLock();

**public** **void** **printOutput**() {

System.out.println("Hello!");

}

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

**if** (myLock.tryLock()) {

myLock.lock();

printOutput();

}**else**

System.out.println("The lock is not accessible.");

}

}

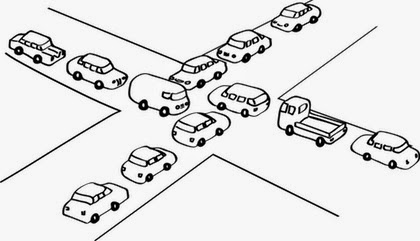
Since here myLock is instance variable, each instance of SimpleLock has their own myLock instance, which means firstThread and secondThread are using different lock and they can run protected code simultaneously.  
  
2) Second mistake Java beginners do is forget to call unlock() method, just like above example. without calling unlock() method, Thread will not release its lock and another thread waiting for that lock will never get that. Nothing will happen in this test program, but once you write this kind of code in real application, you will see nasty issues like deadlock, starvation and data corruption.  By the way Lock interface also offers several advantages over synchronized keyword, check [here](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html) to learn more.  
  
That's all about **how to use Locks in multi-threaded Java program for synchronization**. Let me know if you have any difficult understanding Locks in Java or anything related to multi-threading, Will be glad to help you. For further reading, you can explore Java documentation of Lock interface and it's various implementation classes

Read more: [http://javarevisited.blogspot.com/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html#ixzz3IlOwZdf5](http://javarevisited.blogspot.com/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html" \l "ixzz3IlOwZdf5)

**1)  What is Thread in Java?**  
Thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound task. For example, if one thread takes 100 millisecond to do a job, you can use 10 thread to reduce that task into 10 millisecond. Java provides excellent support for multi-threading at language level, and its also one of strong selling point. For more details [see here](http://java67.blogspot.com/2014/01/10-points-about-thread-and-javalangthread-in-java.html)  
  
  
**2)  Difference between Thread and Process in Java?**  
Thread is subset of Process, in other words one process can contain multiple threads. Two process runs on different memory space, but all threads share same memory space. Don't confuse this with stack memory, which is different for different thread and used to store local data to that thread. For more detail see this [answer](http://java67.blogspot.com/2012/12/what-is-difference-between-thread-vs-process-java.html).  
  
  
**3)  How do you implement Thread in Java?**  
At language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it need a task to execute, which is instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run() method either by extending Thread class or just implementing Runnable interface. For detailed answer and discussion see this [article](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html).  
  
  
**4)  When to use Runnable vs Thread in Java?**  
This is follow-up of previous multi-threading interview question. As we know we can implement thread either by extending Thread class or implementing Runnable interface, question arise, which one is better and when to use one? This question will be easy to answer, if you know that Java programming language doesn't support multiple inheritance of class, but it allows you to implement multiple interface. Which means, its better to implement Runnable than extends Thread, if you also want to extend another class e.g. Canvas or CommandListener. For more points and discussion you can also refer this [post](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html).  
  
  
**6)  Difference between start() and run() method of Thread class?**  
One of trick Java question from early days, but still good enough to differentiate between shallow understanding of Java threading model start() method is used to start newly created thread, while start() internally calls run() method, there is difference calling run() method directly. When you invoke run() as normal method, its called in the same thread, no new thread is started, which is the case when you call start() method. Read this [answer](http://javarevisited.blogspot.sg/2012/03/difference-between-start-and-run-method.html) for much more detailed discussion.  
  
  
**7)  Difference between Runnable and Callable in Java?**  
Both Runnable and Callable represent task which is intended to be executed in separate thread. Runnable is there from JDK 1.0, while Callable was added on JDK 1.5. Main difference between these two is that Callable's call() method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return Future object, which can hold result of computation. See my [blog post](http://java67.blogspot.com/2013/01/difference-between-callable-and-runnable-java.html) on same topic for more in-depth answer of this question.  
  
  
**8)  Difference between CyclicBarrier and CountDownLatch in Java?**  
Though both CyclicBarrier and CountDownLatch wait for number of threads on one or more events, main difference between them is that you can not re-use CountDownLatch once count reaches to zero, but you can reuse same CyclicBarrier even after barrier is broken.  See this [answer](http://javarevisited.blogspot.com/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html) for few more points and sample code example.  
  
  
**9)  What is Java Memory model?**  
Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-before relationship. This relationship defines several rules which allows programmers to anticipate and reason behaviour of concurrent Java programs. For example, happens-before relationship guarantees :

* Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.
* An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.
* A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.
* A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from Thread.join() or by Thread.isAlive() returning false, also known as Thread start rule.
* A thread calling interrupt on another thread happens-before the interrupted thread detects the interrupt( either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.
* The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.
* If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

I strongly suggest to read Chapter 16 of Java Concurrency in Practice to understand Java Memory model in more detail.  
  
  
**10) What is volatile variable in Java?**  
volatile is a special modifier, which can only be used with instance variables. In concurrent Java programs, changes made by multiple threads on instance variables is not visible to other in absence of any synchronizers e.g. synchronized keyword or locks. Volatile variable guarantees that a write will happen before any subsequent read, as stated *"volatile variable rule"* in previous question. Read this [answer](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html) to learn more about volatile variable and when to use them.  
  
  
**11) What is thread-safety? is Vector a thread-safe class?** (Yes, see [details](http://javarevisited.blogspot.sg/2011/09/difference-vector-vs-arraylist-in-java.html))  
Thread-safety is a property of an object or code which guarantees that if executed or used by multiple thread in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modifies state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.  
  
  
**12) What is race condition in Java? Given one example?**  
Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As name suggests, race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between threads. One example of race condition is out-of-order processing, see this [answer](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html) for some more example of race conditions in Java programs.  
  
  
**13) How to stop thread in Java?**  
I always said that Java provides rich APIs for everything but ironically Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. stop(), suspend() and resume() which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads. Programmers mainly rely on the fact that thread stops automatically as soon as they finish execution of run() or call() method. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or interrupt threads to abruptly cancel tasks. See this [tutorial](http://javarevisited.blogspot.com/2011/10/how-to-stop-thread-java-example.html) for sample code of stopping thread in Java.  
  
  
**14) What happens when an Exception occurs in a thread?**  
This is one of the good [tricky Java question](http://java67.blogspot.sg/2012/09/top-10-tricky-java-interview-questions-answers.html) I have seen on interviews. In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its UncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.  
  
  
**15) How do you share data between two thread in Java?**  
You can share data between threads by using shared object, or concurrent data-structure like BlockingQueue. See this tutorial to learn [inter thread communication in Java](http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html). It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.  
  
  
**16) Difference between notify and notifyAll in Java?**  
This is another tricky questions from core Java interviews, since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There notify() method doesn't provide any way to choose a particular thread, that's why its only useful when you know that there is only one thread is waiting. On the other hand, notifyAll() sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further. See my [blog post](http://javarevisited.blogspot.com/2012/10/difference-between-notify-and-notifyall-java-example.html) on similar topic for more detailed answer and code example.  
  
  
**17) Why wait, notify and notifyAll are not inside thread class?**  
This is a design related question, which checks what candidate thinks about existing system or does he ever thought of something which is so common but looks in-appropriate at first. In order to answer this question, you have to give some reasons why it make sense for these three method to be in Object class, and why not on Thread class. One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object. You can also see this [article](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) for more elaborate answer of this question.  
  
  
**18) What is ThreadLocal variable in Java?**  
ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, ThreadLocal variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using ThreadLocal. Since that class is expensive, its not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds in one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and Second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is ThreadLocalRandom class, which reduces number of instances of expensive-to-create Random object in multi-threading environment. See this [answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html) to learn more about thread local variables in Java.  
  
  
**19) What is FutureTask in Java?**  
FutureTask represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A FutureTask object can be used to wrap a Callable or Runnable object. Since FutureTask also implements Runnable, it can be submitted to an Executor for execution.  
  
  
**20) Difference between interrupted and isInterrupted method in Java?**  
Main difference between interrupted() and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the [static method](http://java67.blogspot.com/2012/11/what-is-static-class-variable-method.html) Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted() method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt  
  
  
**21) Why wait and notify method are called from synchronized block?**  
Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A more subtle reason is to avoid race condition between wait and notify calls. To learn more about this, check my similarly titled post [here](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html).  
  
  
**22) Why you should check condition for waiting in a loop?**  
Its possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.  
  
  
**23) Difference between synchronized and concurrent collection in Java?**  
Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like ConcurrentHashMap, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table. See this [answer](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-synchronized.html) for more differences between synchronized and concurrent collection in Java.  
  
  
**24) Difference between Stack and Heap in Java?**  
Why do someone this question as part of multi-threading and concurrency? because Stack is a memory area which is closely associated with threads. To answer this question, both stack and heap are specific memories in Java application. Each thread has their own stack, which is used to store local variables, method parameters and call stack. Variable stored in one Thread's stack is not visible to other. On other hand, heap is a common memory area which is shared by all threads. Objects whether local or at any level is created inside heap. To improve performance thread tends to cache values from heap into their stack, which can create problems if that variable is modified by more than one thread, this is where volatile variables comes in picture. volatile suggest threads to read value of variable always from main memory. See this [article](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html) to learn more about stack and heap in Java to answer this question in greater detail.  
  
  
**25) What is thread pool? Why should you thread pool in Java?**  
Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issue, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of thread) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks). See this [article](http://javarevisited.blogspot.com/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html) to learn more about thread pools in Java to prepare detailed answer of this question.  
  
  
**26) Write code to solve Producer Consumer problem in Java?**  
Most of the threading problem you solved in real world are of category of Producer consumer pattern, where one thread is producing task and other thread is consuming that. You must know how to do inter thread communication to solve this problem. At lowest level, you can use wait and notify to solve this problem, and at high level you can leverage Semaphore or BlockingQueue to implement Producer consumer pattern, as shown in this [tutorial](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html).  
  
  
**27) How do you avoid deadlock in Java? Write Code?**

[](http://4.bp.blogspot.com/-m2IldPcxiJI/U6-Zwvkdd1I/AAAAAAAABns/-zHIHjzM3nM/s1600/deadlock+in+Java.jpg)

Deadlock is a condition in which two threads wait for each other to take action which allows them to move further. It's a serious issue because when it happen your program hangs and doesn't do the task it is intended for. In order for deadlock to happen, following four condition must be true :

* **Mutual Exclusion :** At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
* **Hold and Wait :** A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
* **No Pre-emption :** The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.
* **Circular Wait :** A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

Easiest way to avoid deadlock is to prevent *Circular wai*t, and this can be done by acquiring locks in a particular order and releasing them in reverse order, so that a thread can only proceed to acquire a lock if it held the other one. Check this [tutorial](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html) for actual code example and detailed discussion on techniques of avoiding deadlock in Java.   
  
  
**28) Difference between livelock and deadlock in Java?**  
This question is extension of previous interview question. A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, main difference between livelock and deadlock is that in former state of process change but no progress is made.  
  
  
**29) How do you check if a Thread holds a lock or not?**  
I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object. You can also check this article for more detailed [answer](http://javarevisited.blogspot.com/2010/10/how-to-check-if-thread-has-lock-on.html).  
  
  
**30) How do you take thread dump in Java?**  
There are multiple ways to take thread dump of Java process depending upon operating system. When you take thread dump, JVM dumps state of all threads in log files or standard error console. In windows you can use Ctrl + Break key combination to take thread dump, on Linux you can use kill -3 command for same. You can also use a tool called jstack for taking thread dump, it operate on process id, which can be found using another tool called jps.  
  
  
**31) Which JVM parameter is used to control stack size of thread?**  
This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this [list of JVM options](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html) to learn more about this parameter.  
  
  
**32) Difference between synchronized and ReentrantLock in Java?**  
There were days when only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you can not extend lock beyond a method or block boundary, you can not give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behaviour and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities. See [this article](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html) to learn about those capabilities and some more differences between synchronized vs ReentrantLock in Java.  
  
  
**33) There are three threads T1, T2 and T3? How do you ensure sequence T1, T2, T3 in Java?** Sequencing in multi-threading can be achieved by different means but you can simply use join() method of thread class to start a thread when another one is finished its execution. To ensure three threads execute you need to start the last one first e.g. T3 and then call join methods in reverse order e.g. T3 calls T2. join, and T2 calls T1.join, this ways T1 will finish first and T3 will finish last. To learn more about join method, see this [tutorial](http://javarevisited.blogspot.sg/2013/02/how-to-join-multiple-threads-in-java-example-tutorial.html).  
  
  
**34) What does yield method of Thread class do?**  
Yield method is one way to request current thread to relinquish CPU so that other thread can get chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for same thread to get CPU back and start its execution again. See this [article](http://java67.blogspot.sg/2012/08/difference-between-yield-and-wait.html) to learn more about yield method and to answer this question better.  
  
  
**35) What is concurrence level of ConcurrentHashMap in Java?**  
ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into number of sections. This partitioning is achieved using concurrency level. It's optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. To learn more about concurrency level and internal resizing, see my post [How ConcurrentHashMap works in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html).  
  
  
**36) What is Semaphore in Java?**  
Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect expensive resource which is available in fixed number e.g. database connection in pool. See this [article](http://javarevisited.blogspot.com/2012/05/counting-semaphore-example-in-java-5.html) to learn more about counting Semaphore in Java.  
  
  
**37) What happens if you submit task, when queue of thread pool is already fill?**  
This is another tricky question in my list. Many programmer will think that it will block until a task is cleared but its true. ThreadPoolExecutor's submit() method throws RejectedExecutionException if the task cannot be scheduled for execution.  
  
  
**38) Difference between submit() and execute() method thread pool in Java?**  
Both method are ways to submit task to thread pools but there is slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly it does not return anything. It's return type is void. On other hand submit() is overloaded method, it can take either Runnable or Callable task and can return Future object which can hold pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods. To learn more about thread pools you can check this [article](http://javarevisited.blogspot.sg/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html).  
  
  
**39) What is blocking method in Java?**  
A blocking method is a method which blocks until task is done, for example accept() method of ServerSocket blocks until a client is connected. here blocking means control will not return to caller until task is finished. On the other hand there are asynchronous or non-blocking method which returns even before task is finished. To learn more about blocking method see this [answer](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html).  
  
  
**40) Is Swing thread-safe? What do you mean by Swing thread-safe?**  
You can simply this question as No, Swing is not thread-safe, but you have to explain what you mean by that even if interviewer doesn't ask about it. When we say swing is not thread-safe we usually refer its component, which can not be modified in multiple threads. All update to GUI components has to be done on AWT thread, and Swing provides synchronous and asynchronous callback methods to schedule such updates. You can also read my article to learn more about [swing and thread-safety](http://javarevisited.blogspot.com/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html) to better answer this question. Even next two questions are also related to this concept.  
  
  
**41) Difference between invokeAndWait and invokeLater in Java?**  
These are two methods Swing API provides Java developers to update GUI components from threads other than Event dispatcher thread. InvokeAndWait() synchronously update GUI component, for example a progress bar, once progress is made, bar should also be updated to reflect that change. If progress is tracked in a different thread, it has to call invokeAndWait() to schedule an update of that component by Event dispatcher thread. On other hand, invokeLater() is asynchronous call to update components. You can also refer this [answer](http://javarevisited.blogspot.com/2011/09/invokeandwait-invokelater-swing-example.html) for more points.  
  
  
**42) Which method of Swing API are thread-safe in Java?**  
This question is again related to swing and thread-safety, though components are not thread-safe there are certain method which can be safely call from multiple threads. I know about repaint(), and revalidate() being thread-safe but there are other methods on different swing components e.g. setText() method of JTextComponent, insert() and append() method of JTextArea class.  
  
  
**43) How to create Immutable object in Java?**  
This question might not look related to multi-threading and concurrency, but it is. Immutability helps to simplify already complex concurrent code in Java. Since immutable object can be shared without any synchronization its very dear to Java developers. Core value object, which is meant to be shared among thread should be immutable for performance and simplicity. Unfortunately there is no @Immutable annotation in Java, which can make your object immutable, hard work must be done by Java developers. You need to keep basics like initializing state in constructor, no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object. For step by step guide see my post, [how to make an object Immutable in Java](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html). This will give you enough material to answer this question with confidence.  
  
  
**44) What is ReadWriteLock in Java?**  
In general, read write lock is result of lock stripping technique to improve performance of concurrent applications. In Java, ReadWriteLock is an interface which was added in Java 5 release. A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive. If you want you can implement this interface with your own set of rules, otherwise you can use ReentrantReadWriteLock, which comes along with JDK and supports a maximum of 65535 recursive write locks and 65535 read locks.  
  
  
**45) What is busy spin in multi-threading?**  
Busy spin is a technique which concurrent programmers employ to make a thread wait on certain condition. Unlike traditional methods e.g. wait(), sleep() or yield() which all involves relinquishing CPU control, this method does not relinquish CPU, instead it just runs empty loop. Why would someone do that? to preserve CPU caches. In multi core system, its possible for a paused thread to resume on different core, which means rebuilding cache again. To avoid cost of rebuilding cache, programmer prefer to wait for much smaller time doing busy spin. You can also see this [answer](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html) to learn more about this question.  
  
  
**46) Difference between volatile and atomic variable in Java?**  
This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.  
  
  
**47) What happens if a thread throws an Exception inside synchronized block?**  
This is one more tricky question for average Java programmer, if he can bring the fact about whether lock is released or not is key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reason I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing lock in [finally block](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html).  
  
  
**48) What is double checked locking of Singleton?**  
This is one of the very popular question on Java interviews, and despite its popularity, chances of candidate answering this question satisfactory is only 50%. Half of the time, they failed to write code for double checked locking and half of the time they failed how it was broken and fixed on Java 1.5. This is actually an old way of creating thread-safe singleton, which tries to optimize performance by only locking when Singleton instance is created first time, but because of complexity and the fact it was broken for JDK 1.4,  I personally don't like it. Anyway, even if you not prefer this approach its good to know from interview point of view. Since this question deserve a detailed answer, I have answered in a separate post, you can read my post [how double checked locking on Singleton works](http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html) to learn more about it.  
  
  
**49) How to create thread-safe Singleton in Java?**  
This question is actually follow-up of previous question. If you say you don't like double checked locking then Interviewer is bound to ask about alternative ways of creating thread-safe Singleton class. There are actually man, you can take advantage of class loading and static variable initialization feature of JVM to create instance of Singleton, or you can leverage powerful enumeration type in Java to create Singleton. I actually preferred that way, you can also read this [article](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) to learn more about it and see some sample code.  
  
  
**50) List down 3 multi-threading best practice you follow?**  
This is my favourite question, because I believe that you must follow certain best practices while writing concurrent code which helps in performance, debugging and maintenance. Following are three best practices, I think an average Java programmer should follow :

* **Always give meaningful name to your thread**This goes a long way to find a bug or trace an execution in concurrent code. OrderProcessor, QuoteProcessor or TradeProcessor is much better than Thread-1. Thread-2 and Thread-3. Name should say about task done by that thread. All major framework and even JDK follow this best practice.
* **Avoid locking or Reduce scope of Synchronization**  
  Locking is costly and context switching is even more costlier. Try to avoid synchronization and locking as much as possible and at bare minimum, you should reduce critical section. That's why I prefer synchronized block over synchronized method, because it gives you absolute control on scope of locking.
* **Prefer Synchronizers over wait and notify**  
  Synchronizers like CountDownLatch, Semaphore, CyclicBarrier or Exchanger simplifies coding. It's very difficult to implement complex control flow right using wait and notify. Secondly, these classes are written and maintained by best in business and there is good chance that they are optimized or replaced by better performance code in subsequent JDK releases. By using higher level synchronization utilities, you automatically get all these benefits.
* **Prefer Concurrent Collection over Synchronized Collection**  
  This is another simple best practice which is easy to follow but reap good benefits. Concurrent collection are more scalable than their synchronized counterpart, that's why its better to use them while writing concurrent code. So next time if you need map, think about ConcurrentHashMap before thinking Hashtable. See my article [Concurrent Collections in Java](http://javarevisited.blogspot.com/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html), to learn more about modern collection classes and how to make best use of them.

**51) How do you force start a Thread in Java?**  
This question is like how do you force garbage collection in Java, their is no way, though you can make request using System.gc() but its not guaranteed. On Java multi-threading their is absolute no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.  
  
  
**52) What is fork join framework in Java?**  
The fork join framework, introduced in JDK 7 is a powerful tool available to Java developer to take advantage of multiple processors of modern day servers. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application. One significant advantage of The fork/join framework is that it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy. See this [article](http://javarevisited.blogspot.com/2011/09/fork-join-task-java7-tutorial.html) for much more detailed answer of this question.  
  
  
**53) What is difference between calling wait() and sleep() method in Java multi-threading?**  
Though both wait and sleep introduce some form of pause in Java application, they are tool for different needs. Wait method is used for inter thread communication, it relinquish lock if waiting condition is true and wait for notification when due to action of another thread waiting condition becomes false. On the other hand sleep() method is just to relinquish CPU or stop execution of current thread for specified time duration. Calling sleep method doesn't release the lock held by current thread. You can also take look at this [article](http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html) to answer this question with more details.  
  
  
That's all on this list of **top 50 Java multi-threading and concurrency interview questions**. I have not shared answers of all the questions but provided enough hints and links to explore further and find answers by yourselves. As I said, let me know if you don't find answer of any particular question and I will add answer here. You can use this list to not only to prepare for your core Java and programming interviews but also to check your knowledge about basics of threads, multi-threading, concurrency, design patterns and threading issues like race conditions, deadlock and thread safety problems. My intention is to make this list of question as mother of all list of Java Multi-threading questions, but this can not be done without your help. You can also share any question with us, which has been asked to you or any question for which you yet to find an answer. This master list is equally useful to Java developers of all levels of experience. You can read through this list even if you have 2 to 3 years of working experience as junior developer or 5 to 6 years as senior developer. It's even useful for freshers and beginners to expand their knowledge. I will add new and latest multi-threading question as and when I come across, and I request you all to ask, share and answer questions via comments to keep this list relevant for all Java programmers.

Read more: <http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html#ixzz3IlPk7L00>

**Q1)If a class has a synchronised method and non-synchronised method, can multiple threads execute the non-synchronised methods?**

Ans)Yes. If a class has a synchronised and non-synchronised methods, multiple threads can access the non-synchronised methods.

**Q2)If a thread goes to sleep does it hold the lock?**

Ans) Yes when a thread goes to sleep it does not release the lock.

**Q3)Can a thread hold multiple locks at the same time?**

Ans) Yes. A thread can hold multiple locks at the same time. Once a thread acquires a lock and enters into the synchronized method / block, it may call another synchronized method and acquire a lock on another object.

**Q4) Can a thread call multiple synchronized methods on the object of which it hold the lock?**

Ans) Yes. Once a thread acquires a lock in some object, it may call any other synchronized method of that same object using the lock that it already holds.

**Q5) Can static methods be synchronized?**

Ans) Yes. As static methods are class methods and have only one copy of static data for the class, only one lock for the entire class is required. Every class in java is represented by java.lang.Class instance. The lock on this instance is used to synchronize the static methods.

**Q6) Can two threads call two different static synchronized methods of the same class?**

Ans) No. The static synchronized methods of the same class always block each other as only one lock per class exists.So no two static synchronized methods can execute at the same time.

**Q7)Does a static synchronized method block a non-static synchronized method?**

Ans)No As the thread executing the static synchronized method holds a lock on the class and the thread executing the non-satic synchronized method holds the lock on the object on which the method has been called, these two locks are different and these threads do not block each other.

**Q8) Once a thread has been started can it be started again?**

Ans) No. Only a thread can be started only once in its lifetime. If you try starting a thread which has been already started once an IllegalThreadStateException is thrown, which is a runtime exception. A thread in runnable state or a dead thread can not be restarted.

**Q9) When does deadlock occur and how to avoid it?**

Ans) When a locked object tries to access a locked object which is trying to access the first locked object. When the threads are waiting for each other to release the lock on a particular object, deadlock occurs .

**Q10) What is a better way of creating multithreaded application? Extending Thread class or implementing Runnable?**

Ans) If a class is made to extend the thread class to have a multithreaded application then this subclass of Thread can not extend any other class and the required application will have to be added to this class as it can not be inherited from any other class. If a class is made to implement Runnable interface, then the class can extend other class or implement other interface.

**Q11) Can the start() method of the Thread class be overridden? If yes should it be overridden?**

Ans) Yes the start() method can be overridden. But it should not be overridden as itâ€™s implementation in thread class has the code to create a new executable thread and is specialised.

**Q12) What are the methods of the thread class used to schedule the threads?**

Ans) The methods are as follows:

* public static void sleep(long millis) throws InterruptedException
* public static void yield()
* public final void join() throws InterruptedException
* public final void setPriority(int priority)
* public final void wait() throws InterruptedException
* public final void notify()
* public final void notifyAll()

**Q13) Which thread related methods are available in Object class?**

Ans) The methods are:

* public final void wait() throws Interrupted exception
* public final void notify()
* public final void notifyAll()

**Q14) Which thread related methods are available in Thread class?**

Ans) Methods which are mainly used :

* public static void sleep(long millis) throws Interrupted exception
* public static void yield() public final void join() throws Interrupted exception
* public final void setPriority(int priority)
* public void start()
* public void interrupt()
* public final void join()
* public void run()
* public void resume()

**Q15) List the methods which when called the thread does not release the locks held?**

Ans) Following are the methods.

* notify()
* join()
* sleep()
* yield()

**Q16) List the methods which when called on the object the thread releases the locks held on that object?**

Ans) wait()

**Q17) Does each thread has its own thread stack?**

Ans) Yes each thread has its own call stack. For eg

Thread t1 = new Thread();

Thread t2 = new Thread();

Thread t3 = t1;

In the above example t1 and t3 will have the same stack and t2 will have its own independent stack.

**Q18) What is thread starvation?**

Ans)In a multi-threaded environment thread starvation occurs if a low priority thread is not able to run or get a lock on the resoruce because of presence of many high priority threads. This is mainly possible by setting thread priorities inappropriately.

**Q19) What is threadLocal variable?**

Ans) ThreadLocal is a class. If a variable is declared as threadLocal then each thread will have a its own copy of variable and would not interfere with the other's thread copy. Typical scenario to use this would be giving JDBc connection to each thread so that there is no conflict.

**ThreadLocal class by JAVA API**

public class ThreadLocal {

public Object get();

public void set(Object newValue);

public Object initialValue();

}

**Implementation of ThreadLocal**

public class ConnectionDispenser {

private static class ThreadLocalConnection extends ThreadLocal {

public Object initialValue() {

return DriverManager.getConnection(ConfigurationSingleton.getDbUrl());

}

}

private static ThreadLocalConnection conn = new ThreadLocalConnection();

public static Connection getConnection() {

return (Connection) conn.get();

}

}