**21.Deep copy and shallow copy?**

A clone is an exact copy of the original. In java, it essentially means the ability to create an object with similar state as the original object. The clone() method provides this functionality.

**Why use clone() method ?**

**The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.**

**Shallow Clonning**

In Java, when the cloning process is done by invoking the clone() method it is called Shallow Cloning. It is the default cloning process in Java where a shallow copy of the original object will be created with exact field.

if you change the value of the cloned objects then it will be reflected in the original as well. Thus, shallow cloning is dependent on the original object.

**Deep Cloning in Java**

In Java, when the cloning process is done by implementing the Cloneable interface it is called Deep Cloning.

In this type of cloning, an exact copy of all the fields of the original object will be created.

But in case, the original object has references to other objects as fields then a copy of those objects will also be created by calling the clone() method. This makes the cloned object independent of the original object and any changes made in any of the object won’t be reflected on the other.

**22.What is synchronization? Object level locking and class level locking?**

***Synchronization***refers to multi-threading. A synchronized block of code can only be executed by one thread at a time. Java supports multiple threads to be executed. This may cause two or more threads to access the same fields or objects.

Synchronization is a process which keeps all concurrent threads in execution to be in synch. Synchronization avoids memory consistence errors caused due to inconsistent view of shared memory.

When a method is declared as synchronized; the thread holds the lock/monitor for that method’s object If another thread is executing the synchronized method, your thread is blocked until that thread releases the monitor.

Synchronization in java is achieved using synchronized keyword. You can use synchronized keyword in your class on defined methods or blocks. Keyword can not be used with variables or attributes in class definition.

**23.Difference between sleep() and wait()?**

sleep() is a method which is used to hold the process for few seconds or the time you wanted but in case of wait() method thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

**The major difference** is that wait() releases the lock or monitor while sleep() doesn’t releases any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.

1) wait is called from synchronized context only while sleep can be called without synchronized block. see [Why to wait and notify needs to call from the synchronized method](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html) for more detail.  
  
2) waiting thread can be awake by calling notify and notifyAll while sleeping thread can not be awakened by calling notify method.  
  
3) wait is normally done on the condition, Thread waits until a condition is true while sleep is just to put your thread on sleep.

4) wait for release lock on an object while waiting while sleep doesn’t release the lock while waiting.  
  
5) The wait() method is called on an object on which the synchronized block is locked, while sleep is called on the Thread.

**24.Can you assign null to this reference variable?**

NO. You can’t. In java, left hand side of an assignment statement must be a variable. ‘this’ is a special keyword which represent the current instance always. This is not any variable.

Similarly, null can not be assigned to ‘super’ or any such keyword for that matter.

**25.What if the difference between && and &??**

& is bitwise and && is logical.

* & evaluates both sides of the operation.
* && evaluates the left side of the operation, if it’s true, it continues and evaluates the right side.

**26.How to override equals and hashCode() methods?**

hashCode() and equals() methods have been defined in Object class which is parent class for java objects. For this reason, all java objects inherit a default implementation of these methods.

hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable like data structure. By default, Object’s hashCode() method returns and integer representation of memory address where object is stored.

equals() method, as name suggest, is used to simply verify the equality of two objects.  Default implementation simply check the object references of two objects to verify their equality.

Below are the important points to keep remember while overriding these functions.

1. Always use same attributes of an object to generate hashCode() and equals() both. As in our case, we have used employee id.
2. equals() must be consistent (if the objects are not modified, then it must keep returning the same value).
3. Whenever a.equals(b), then a.hashCode() must be same as b.hashCode().
4. If you override one, then you should override the other.

[Read more interesting facts and how to guide here.](https://howtodoinjava.com/java/related-concepts/working-with-hashcode-and-equals-methods-in-java/)

**27.Explain all access modifiers?**

**28.What is garbage collection? Can we enforce it?**

Garbage collection is an automatic memory management feature in many modern programming languages, such as Java and languages in the .NET framework. Languages that use garbage collection are often interpreted or run within a virtual machine like the JVM. In each case, the environment that runs the code is also responsible for garbage collection. A GC has two goals: any unused memory should be freed, and no memory should be freed unless the program will not use it anymore.

Can you force garbage collection?? Nope, System.gc() is as close as you can get. Your best option is to call System.gc() which simply is a hint to the garbage collector that you want it to do a collection. There is no way to force and immediate collection though as the garbage collector is non-deterministic. Also, under the documentation for OutOfMemoryError it declares that it will not be thrown unless the VM has failed to reclaim memory following a full garbage collection. So if you keep allocating memory until you get the error, you will have already forced a full garbage collection.

[Read more about garbage collection here.](https://howtodoinjava.com/java/garbage-collection/revisiting-memory-management-and-garbage-collection-mechanisms-in-java/)

**29.What is native keyword? Explain in detail?**

The native keyword is applied to a method to indicate that the method is implemented in native code using JNI. It marks a method, that it will be implemented in other languages, not in Java.

Native methods were used in the past to write performance critical sections but with Java getting faster this is now less common. Native methods are currently needed when

* You need to call a library from Java that is written in other language.
* You need to access system or hardware resources that are only reachable from the other language (typically C). Actually, many system functions that interact with real computer (disk and network IO, for instance) can only do this because they call native code.

The downsides of using native code libraries are also significant:

1. JNI / JNA have a tendency to destabilize the JVM, especially if you try to do something complicated. If your native code gets native code memory management wrong, there’s a chance that you will crash the JVM. If your native code is non-reentrant and gets called from more than one Java thread, bad things will happen … sporadically. And so on.
2. Java with native code is harder to debug than pure Java or pure C/C++.
3. Native code can introduce significant platform dependencies / issues for an otherwise platform independent Java app.
4. Native code requires a separate build framework, and that may have platform / portability issues as well.

**30.What is serialization? Explain the catches?**

In computer science, in the context of data storage and transmission, serialization is the process of translating data structures or object state into a format that can be stored  and “resurrected” later in the same or another computer environment.  When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object.

Java provides automatic serialization which requires that the object be marked by implementing the java.io.Serializable interface. Implementing the interface marks the class as “okay to serialize,” and Java then handles serialization internally. There are no serialization methods defined on the Serializable interface, but a serializable class can optionally define methods with certain special names and signatures that if defined, will be called as part of the serialization/deserialization process.

Once an object is serialized, changes in its class break the de-serialization process. To identify the future changes in your class which will be compatible and others which will prove incompatible, please read the[**full guide here**](https://howtodoinjava.com/java/serialization/a-mini-guide-for-implementing-serializable-interface-in-java/). In short, I am listing down here:

**Incompatible changes**

* Deleting fields
* Moving classes up or down the hierarchy
* Changing a non-static field to static or a non-transient field to transient
* Changing the declared type of a primitive field
* Changing the writeObject or readObject method so that it no longer writes or reads the default field data
* Changing a class from Serializable to Externalizable or vice-versa
* Changing a class from a non-enum type to an enum type or vice versa
* Removing either Serializable or Externalizable
* Adding the writeReplace or readResolve method to a class

**Compatible changes**

* Adding fields
* Adding/ Removing classes
* Adding writeObject/readObject methods [defaultReadObject or defaultWriteObject should be called first]
* Removing writeObject/readObject methods
* Adding java.io.Serializable
* Changing the access to a field
* Changing a field from static to non-static or transient to non transient

Happy Learning !!