**Transient keyword in Java**

**transient** is a variables modifier used in [serialization](http://quiz.geeksforgeeks.org/serialization-in-java/). At the time of serialization, if we don’t want to save value of a particular variable in a file, then we use **transient** keyword.

When JVM comes across **transient** keyword, it ignores original value of the variable and save default value of that variable data type.

**transient** keyword plays an important role to meet security constraints. There are various real-life examples where we don’t want to save private data in file.

Another use of **transient** keyword is not to serialize the variable whose value can be calculated/derived using other serialized objects or system such as age of a person, current date, etc.

Practically we serialized only those fields which represent a state of instance, after all serialization is all about to save state of an object to a file.

It is good habit to use **transient** keyword with private confidential fields of a class during serialization.

**transient and static :** Since **static** fields are not part of state of the object, there is no use/impact of using **transient** keyword with static variables. However there is no compilation error.

**transient and final :** final variables are directly serialized by their values, so there is no use/impact of declaring final variable as **transient**. There is no compile-time error though

1. **Instance Variables:** These variables are serialized, so during deserialization we will get back the serialized state.
2. **Static Variables:** These variables are not serialized, So during deserialization static variable value will loaded from the class.(Current value will be loaded.)
3. **transient Variables:** transient variables are not serialized, so during deserialization those variables will be initialized with corresponding default values (ex: for objects null, int 0).
4. **Super class variables:** If super class also implemented Serializable interface then those variables will be serialized, otherwise it won't serialize the super class variables. and while deserializing, JVM will run default constructor in super class and populates the default values. Same thing will happen for all superclasses.

**public** **class** Test **implements** Serializable {

// Normal variables

**int** i = 10, j = 20;

// Transient variables

**transient** **int** k = 30;

// Use of transient has no impact here

**transient** **static** **int** *l* = 40;

**transient** **final** **int** m = 50;

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

{

Test input = **new** Test();

// serialization

FileOutputStream fos = **new** FileOutputStream("abc.txt");

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(input);

// de-serialization

FileInputStream fis = **new** FileInputStream("abc.txt");

ObjectInputStream ois = **new** ObjectInputStream(fis);

Test output = (Test)ois.readObject();

System.***out***.println("i = " + output.i);

System.***out***.println("j = " + output.j);

System.***out***.println("k = " + output.k);

System.***out***.println("l = " + output.*l*);

System.***out***.println("m = " + output.m);

}

}

Output:

i = 10

j = 20

k = 0

l = 40

m = 50

**Volatile Keyword in Java**

If the variable keeps on changing such type of variables we have to declare with volatile modifier.

Volatile is a modifier applicable only for variables but not for method and class.

If a variable declared as volatile then for every thread a separate local copy will be created.

Every intermediate modification performed by that thread will takes place in local copy instead of master copy.

Once the value got finalized just before terminating the thread the master copy value will be updated with local stable value.

**Advantage of Volatile**

The main advantage of Volatile keyword is we can resolve data inconsistency problems.

**Dis-Advantage of Volatile**

The main dis-advantage of Volatile keyword is, creating and maintaining a separate copy for every thread, increases complexity of the programming and effects performance of the system.

Hence if there is no specific requirement it is never recommended to use volatile keyword and it is almost outdated keyword.

**Note:** Volatile variable means its value keeps on changing where as final variable means its value never changes. Hence final-Volatile combination is illegal combination for variables.

**Difference between synchronized and volatile keyword in Java**

What is the difference between volatile and synchronized is another popular [core Java question](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html) asked on multi-threading and concurrency interviews. Remember volatile is not a replacement of synchronized keyword but can be used as an alternative in certain cases. Here are few differences between volatile and synchronized keyword in Java.  
  
1. The volatile keyword in Java is a field modifier while synchronized modifies code blocks and methods.  
  
2. Synchronized obtains and releases the lock on monitor’s Java volatile keyword doesn't require that.  
  
3. Threads in Java can be blocked for waiting for any monitor in case of synchronized, that is not the case with the [volatile keyword](http://java67.blogspot.com/2012/11/difference-between-transient-vs-volatile-modifier-variable-java.html) in Java.  
  
4. Synchronized method affects performance more than a volatile keyword in Java.  
  
5. Since volatile keyword in Java only synchronizes the value of one variable between Thread memory and "main" memory while synchronized synchronizes the value of all variable between thread memory and "main" memory and locks and releases a monitor to boot. Due to this reason synchronized keyword in Java is likely to have more overhead than volatile.  
  
6. You cannot synchronize on the null object but your volatile variable in Java could be null.  
  
7. From Java 5 writing into a volatile field has the same memory effect as a monitor release, and reading from a volatile field has the same memory effect as a monitor acquire  
  
  
In short, volatile keyword in Java is not a replacement of synchronized block or method but in some situation is very handy and can save performance overhead which comes with use of synchronization in Java.

Read more: <https://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html#ixzz5gmP6eAcl>