**Why all classes inherited from Object**

It is not an explicit requirement forced on the developer. If a class is declared without extending another class then it will implicitly extend [Object](http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) class. This is taken care of by the JVM. This is not applicable for the interfaces. A Java interface does not extends the Object.

Following could be the reasons for this design decision,

* By having the Object as the super class of all Java classes, without knowing the type we can pass around objects using the Object declaration.
* Before generics was introduced, imagine the state of heterogeneous Java collections. A collection class like ArrayList allows to store any type of classes. It was made possible only by Object class hierarchy.
* The other reason would be to bring a common blueprint for all classes and have some list of functions same among them. I am referring to methods like [hashCode()](https://javapapers.com/core-java/hashcode-and-equals-methods-override/), clone(), toString() and methods for threading which is defined in Object class.

**Data Structures**

A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently. efficient data structures are a key to designing efficient algorithms.

**OutOfMemory Vs StackOverFlow**

When you start JVM you define how much RAM it can use use for processing. JVM divides this into certain memory locations for its processing purpose, two of those are Stack & Heap

OutOfMemoryError is related to Heap. If you have large objects (or) referenced objects in memory, then you will see OutofMemoryError. If you have strong references to objects, then GC can't clean the memory space allocated for that object. When JVM tries to allocate memory for new object and not enough space available it throws OutofMemoryError because it can't allocate required amount of memory.

**How to avoid**: Make sure un-necessary objects are available for GC

StackOverflowError is related to stack. All your local variables and methods calls related data will be on stack. For every method call one stack frame will be created and local as well as method call related data will be placed inside the stack frame. Once method execution is completed, stack frame will be removed. ONE WAY to reproduce this is, have infinite loop for method call, you will see stackoverflow error, because stack frame will be populated with method data for every call but it won't be freed (removed).

**How to avoid**: Make sure method calls are ending (not in infinite loop)

StackOverflowError happens when you execute too many methods one inside another (for example with an infinite recursion), which is limited by the size of the stack.

OutOfMemoryError happens when the JVM runs out of space to allocate new objects, which are allocated on the heap

Imagine you have a function like the following

public void f(int x) {

return f(x + 1);

}

When you'll call it the call will call f again and again and again. At each call a bit of information is stored on the stack. Since the stack is limited in size you will get a StackOverflowError.

Now imagine the following code:

for (int i = 1; i > 0; i++)

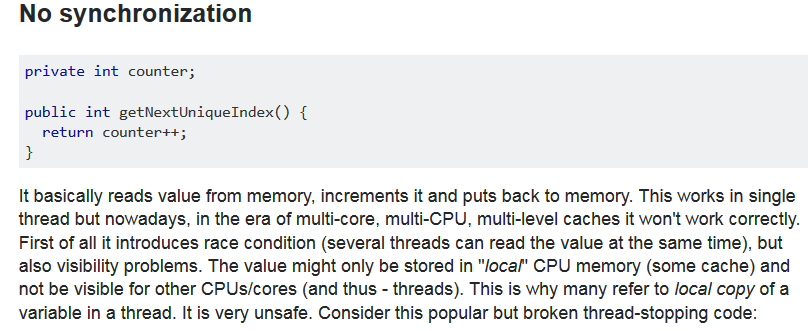
vector.add(new BigObject());

where BigObject is a normal Java object. As you see, the loop never terminates. Each allocation is done on the heap thus it will be filled with BigObjects and you will get an OutOfMemoryError.

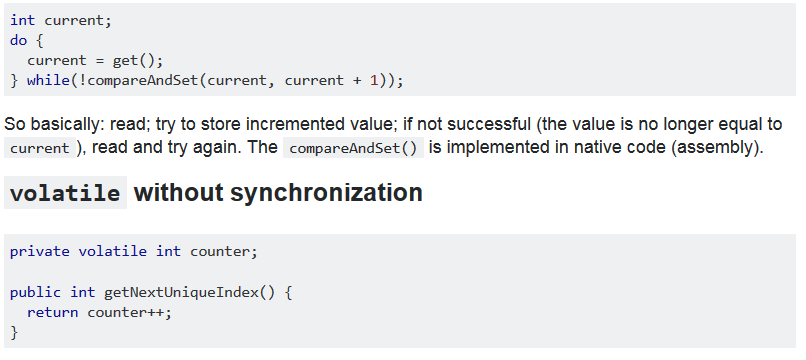
To recap:

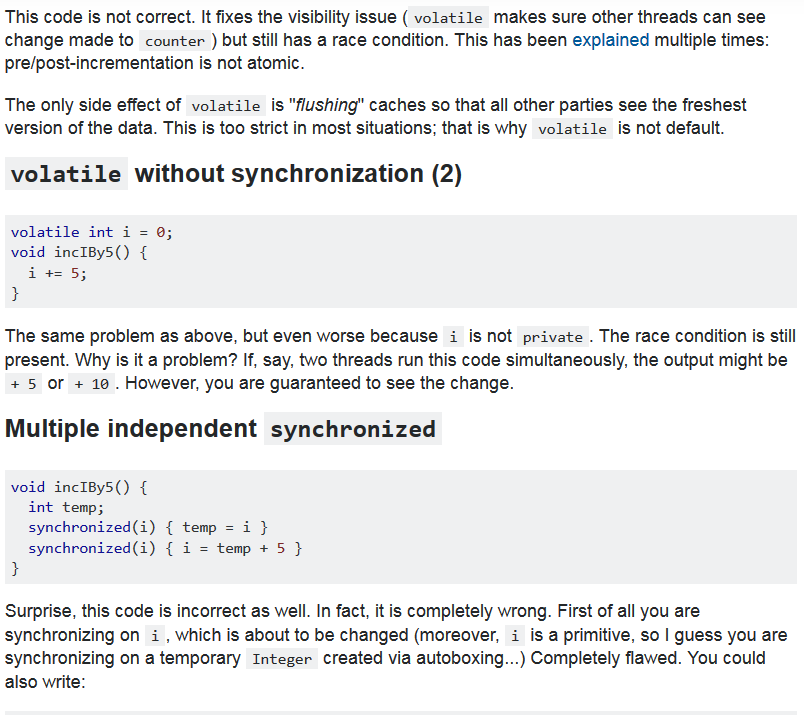
* OutOfMemoryError is thrown when you are creating objects
* StackOverflowError is thrown when you are calling functions

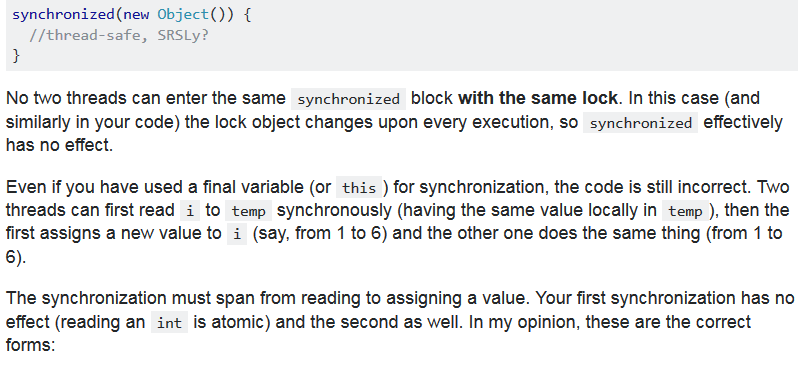
**AtomicInteger Vs Volatile Vs Synchronized**



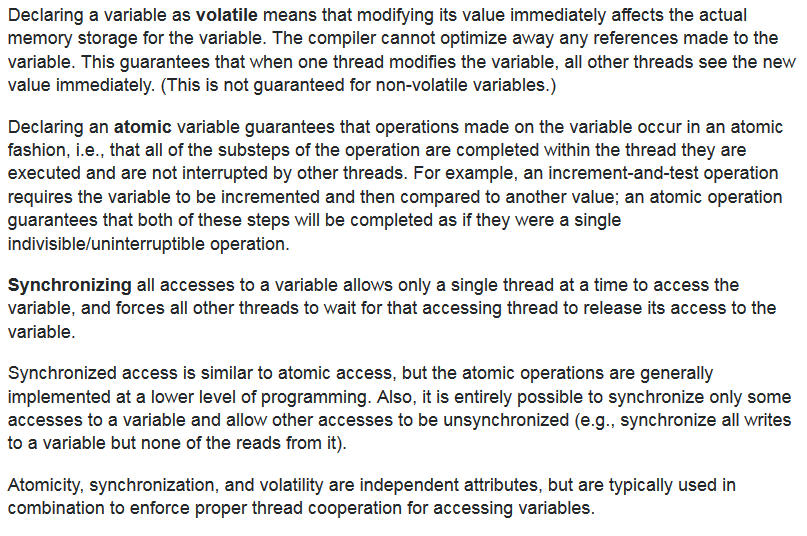












**How to stop a thread in Java**

Either use interrupt or use volatile variable flag inside run method to see when we can exit the

# [difference between strong/soft/weak/phantom reference](https://stackoverflow.com/questions/9809074/java-difference-between-strong-soft-weak-phantom-reference)

Java provides two different types/classes of Reference Objects: **strong** and **weak**. Weak Reference Objects can be further divided into soft and phantom. Let's go point by point.

**Strong Reference Object**

StringBuilder builder = new StringBuilder();

This is the default type/class of Reference Object, if not differently specified: builder is a strong Reference Object. This kind of reference makes the referenced object not eligible for GC. That is, whenever an object is referenced by a chain of strong Reference Objects, it cannot be garbage collected.

**Weak Reference Object**

WeakReference<StringBuilder> weakBuilder = new WeakReference<StringBuilder>(builder);

Weak Reference Objects are not the default type/class of Reference Object and to be used they should be explicitly specified like in the above example. This kind of reference makes the reference object eligible for GC. That is, in case the only reference reachable for the StringBuilder object in memory is, actually, the weak reference, then the GC is allowed to garbage collect the StringBuilder object. When an object in memory is reachable only by Weak Reference Objects, it becomes automatically eligible for GC.

**Levels of Weakness**

Two different levels of weakness can be enlisted: soft and phantom.

A soft Reference Object is basically a weak Reference Object that remains in memory a bit more: normally, it resists GC cycle until memory is available and there is no risk of OutOfMemoryError (in that case, it can be removed).

On the other hand, a phantom Reference Object is useful only to know exactly when an object has been effectively removed from memory: normally they are used to fix weird finalize() revival/resurrection behavior, since they actually do not return the object itself but only help [in keeping track of their memory presence](https://weblogs.java.net/blog/2006/05/04/understanding-weak-references).

Weak Reference Objects are ideal to implement cache modules. In fact, a sort of automatic eviction can be implemented by allowing the GC to clean up memory areas whenever objects/values are no longer reachable by strong references chain. An example is the [WeakHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html) retaining weak keys.

* [**SoftReference**](https://docs.oracle.com/javase/8/docs/api/java/lang/ref/SoftReference.html):Soft reference objects are cleared at the discretion of the garbage collector in response to memory demand. Soft references are most often used to implement memory-sensitive caches. All soft references to softly reachable objects are guaranteed to have been cleared before the virtual machine throws an OutOfMemoryError.
* [**WeakReference**](https://docs.oracle.com/javase/8/docs/api/java/lang/ref/WeakReference.html): Weak reference objects do not prevent their referents from being made finalizable, finalized, and then reclaimed. Weak references are most often used to implement canonicalizing mappings. (Here, Canonicalizing mappings means mapping only reachable object instances.)
* [**PhantomReference**](https://docs.oracle.com/javase/8/docs/api/java/lang/ref/PhantomReference.html): Phantom reference objects are enqueued after the collector determines that their referents may otherwise be reclaimed. Phantom references are most often used for scheduling pre-mortem cleanup actions in a more flexible way than is possible with the Java finalization mechanism. Unlike soft and weak references, phantom references are not automatically cleared by the garbage collector as they are enqueued. An object that is reachable via phantom references will remain so until all such references are cleared or themselves become unreachable.

So in brief: Soft references try to keep the reference. Weak references don’t try to keep the reference. Phantom references don’t free the reference until cleared.

To reuse (and stretch) our restaurant metaphor one last time: A SoftReference is like a customer that says, "I’ll leave my table only when there are no other tables available." A WeakReference is like someone ready to leave as soon as a new customer arrives. A PhantomReference is like someone ready to leave as soon as a new customer arrives, but actually not leaving until the manager gives them permission.