Static Vs Volatile variables in Multi-threading

We all know that Static variables and methods will be associated with the class, rather than with any object. Every instance of the class shares a class variable (static), which is in one fixed location in memory. Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class.

In that case how Static differs from Volatile under Multi-threading?

Declaring static variable means there will be only copy associated with class it doesn't matter how many object get created for that class. The static variable will be accessible even with no Objects created at all. Thread may have locally cached values of it.

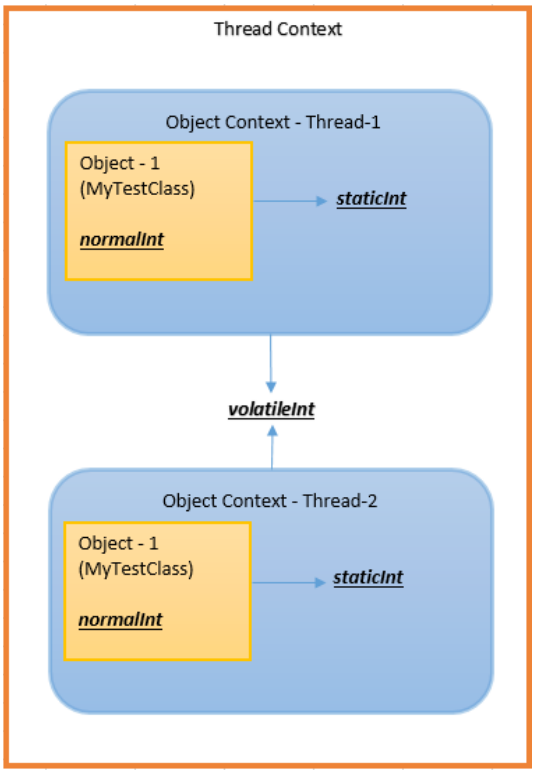
This means that if two threads update a variable of the same Object concurrently, and the variable is not declared volatile, there could be a case in which one of the threads has in cache an old value. Even if you access a static value through multiple threads, each thread can have its local cached copy!

But a volatile variable will keep only one copy in memory and shared across the threads. Lets assume a class with static and volatile variables, with 2 threads accessing it.

```
public class MyTestClass {

 int normalInt;
 static int staticInt;
 volatile int volatileInt;

}
```



```
Class Test {
  int nonStaticInt;
  static int staticInt;
  volatile static int volStaInt;
}
```