1. If a variable is shared between multiple threads you need to use volatile keyword with that variable.  
  
2. If you want to share any variable in which read and write operation is atomic by implementation e.g. read and write in an int or a boolean variable then  you can declare them as volatile variable.

**3. The Java volatile keyword cannot be used with method or class and it can only be used with a primitive type variable or Object.**4. Volatile keyword in Java guarantees that value of the volatile variable will always be read from **main memory** and not from Thread's local **cache.**  
  
If a variable is shared between multiple threads  
  
/\*\*

\* Java program to demonstrate where to use Volatile keyword in Java.

\* In this example Singleton Instance is declared as volatile variable to ensure

\* every thread see updated value for \_instance.

\*

\* @author Javin Paul

\*/

**public** **class** **Singleton**{

**private** **static** **volatile** Singleton \_instance; //volatile variable

**public** **static** Singleton **getInstance**(){

**if**(\_instance == **null**){

**synchronized**(Singleton.class){

**if**(\_instance == **null**)

\_instance = **new** Singleton();

}

}

**return** \_instance;

}

If you look at the code carefully you will be able to figure out:  
1) We are only creating instance one time  
2) We are creating instance lazily at the time of the first request comes.