**Singleton with 2 WAR's**

I was asked a brilliant question by one of our colleagues and here is the question :   
  
"If you have a singleton class which is deployed in two different WAR files, then how many instances will be created?", According to me, each application would have its own set of clients and 2 instances will be created for 2 WAR's, But then the question was changed,   
  
"So if you are still getting 2 instances in 2 WAR's, what if you still want to create only 1 instance irrespective of the WAR's in which the same class is deployed. It may happen that tomorrow the same class would be deployed in another 2 WAR files".   
  
P.S : The Application server and JVM is same for both the WAR files.

This is a class loading issue. Since the two .wars are deployed in the same container, they'll each be provided with their own class loader but they both inherit from a common one.   
If you want to keep a single instance of your singleton class for all the components you deploy, you should include that in the common/parent class path and exclude it from your .war/WEB-INF/lib.   
In tomcat, you should put the jar containing your singleton in your ${TOMCAT\_HOME}/lib (or common/lib, shared/lib) directory so that it will get inherited by everyone.

Note that if you have the "Singleton" class in each war, you will not only get two instances of the singleton, but also two distinct classes: you'll get a [ClassCastException](http://docs.oracle.com/javase/7/docs/api/java/lang/ClassCastException.html) when trying to pass it from one .war to the other, even if it's apparently the same class -- it's not, since a class definition is relative to its class loader.

Well, strictly this will depend on how the classloaders in the container are configured, since this can be changed (depending on the container).   
  
If your Singleton is implemented to use a specific resource to track instances (e.g. a lock file, or is bound in JNDI or a table in a database) you can still have one instance. If you *really* need one instance this would be the container agnostic way of doing it.