1. **PURPOSE OF THE PROJECT**

The project Address Cache Management is created with a motive of creating a central repository called Address Cache Which stores addresses.

A scheduler has been assigned to clear the cache for objects which are exceeding a predefined age/limit.

The cache can be accessed to put elements, remove elements, take the recently added element and in turn remove it from the cache.

The cache will be used for concurrent access as in a multithreading environment.

1. **TECHNICAL DESIGN** 
   1. **DESIGN USED**

The code has been kept very simple but more effort has been made to choose the right implementation so that parallel processing can be done very easily without any hassles.

The Address cache has been made as a **SINGLETON** in JAVA (Singleton in java implies to the fact that at any point of time, any number of objects accessing the cache will get the same instance throughout. The cache instance for all threads remain the same).

The Address cache is extending **a generic concurrent cache**. This has been made keeping in mind that in case the user wants to change the Data type or Object Type of the cache, it can be done easily. The type of values the cache will store is not hardcoded and given an interface architecture for future implementations.

I have chosen the cache to be a map – **CONCURRENT HASH MAP**

Reasons for choosing it as a concurrent hash map

1. Concurrent hash map guarantees thread safety.
2. Functioning of Concurrent hash map –

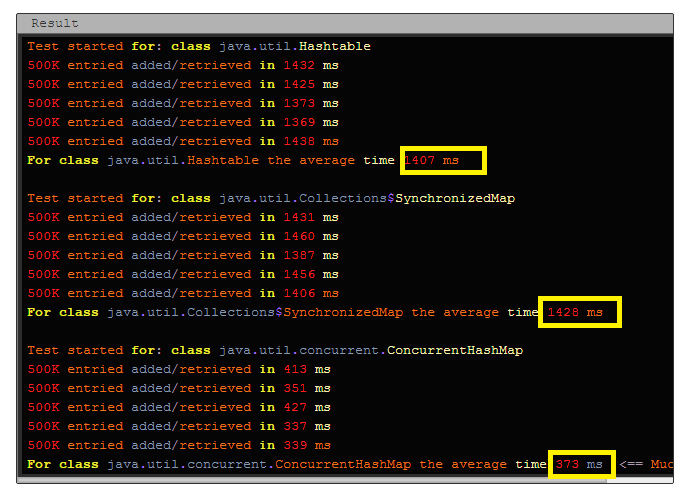
A Concurrent Hash Map is divided into number of segments, for example i use default as 32 on initialization.

A Concurrent Hash Map is internally divided in segments of size 32, so at max 32 threads can work at a time. It means each thread can work on each segment during high concurrency and atmost 32 threads can operate at max which simply maintains 32 locks to guard each bucket of the Concurrent Hash Map.The internal synchronization is done by the Concurrent Hash map itself.

1. Other data structures also provide thread safety for example, synchronized methods or blocks, Collections.synchronizedMap method. But the performance level of Concurrent hash map is better. The time taken by hashmap is lesser as compared to other data structures.

The cache object is a map with key as Inet Address and value as the timestamp.

The concurrent hash map in itself is efficient enough to handle multiple threads with efficiency in performance. The screenshot shows the efficiency of using concurrent hash map with multiple threads(500k threads).



**CLEAR CACHE** – I have set a java timer or scheduler (the time is user defined). The scheduler runs at particular duration, let’s say every hour and checks for each object if the timestamp has exceeded the max time (the max time is user defined). If the object in cache has exceeded the max time, it will remove the element from the cache. The timer is set on server start up.

PUT, REMOVE, PEEK, TAKE OPERATIONS – All these operations are performed by rest services (The url’s to perform each operation has been given).

**PUT**

http://localhost:7001 /AddressCacheManagement/rest/CacheService/put

Method – GET, parameters – address (?address=10.10.10.10)

**REMOVE**

http://localhost:7001/AddressCacheManagement/rest/CacheService/remove Method – GET, parameters – address (?address=10.10.10.10)

**PEEK**

<http://localhost:7001/AddressCacheManagement/rest/CacheService/peek>

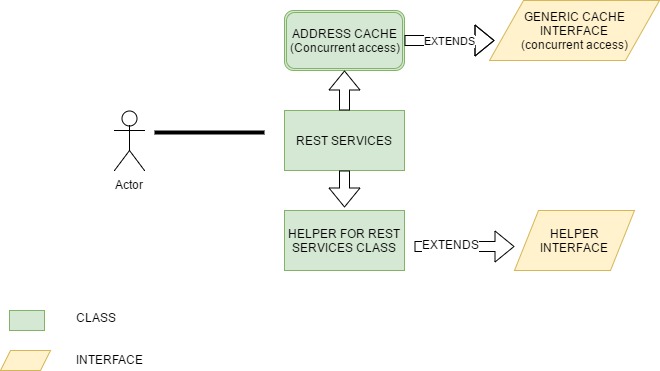
Method – GET

**TAKE**

http://localhost:7001/AddressCacheManagement/rest/CacheService/take

Method – GET

* 1. **DESIGN DIAGRAM**

****

1. **DESIGN – CONCURRENCY AND CORRECTNESS**

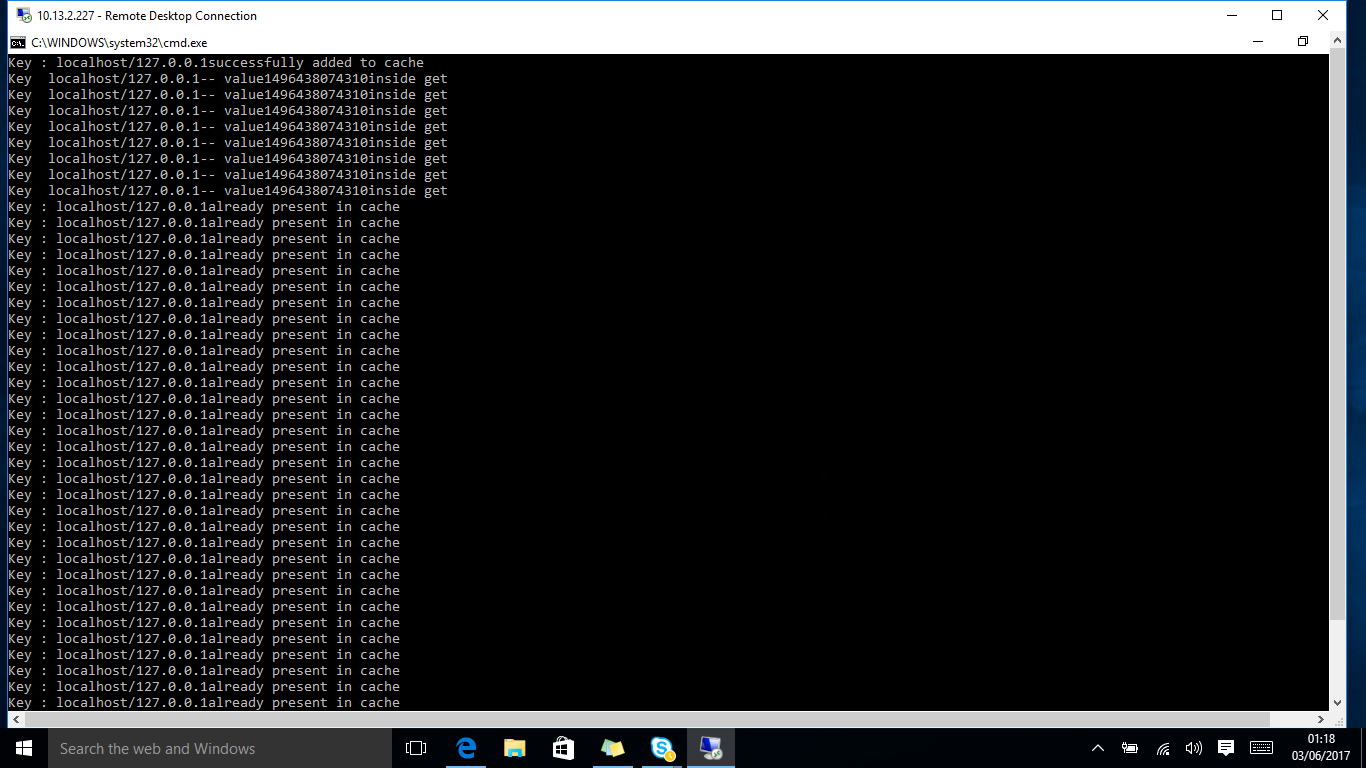
**Concurrency** - Concurrency has been kept in mind by implementing it as a singleton and as a concurrent hash map.

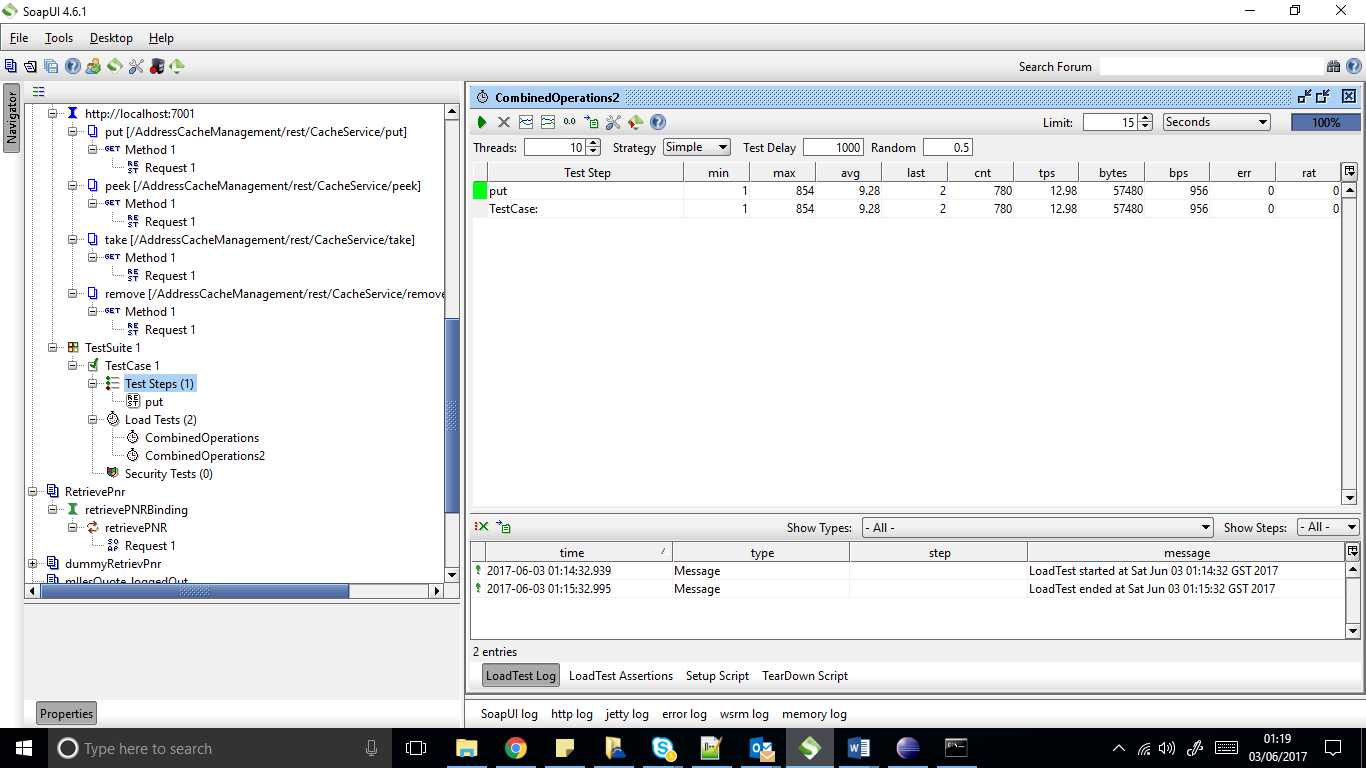
The concurrency has been tested by using SOAP UI application.

Each method (put, remove, peek, take) has been load tested by running multiple threads in SOAPUI framework as a test suite

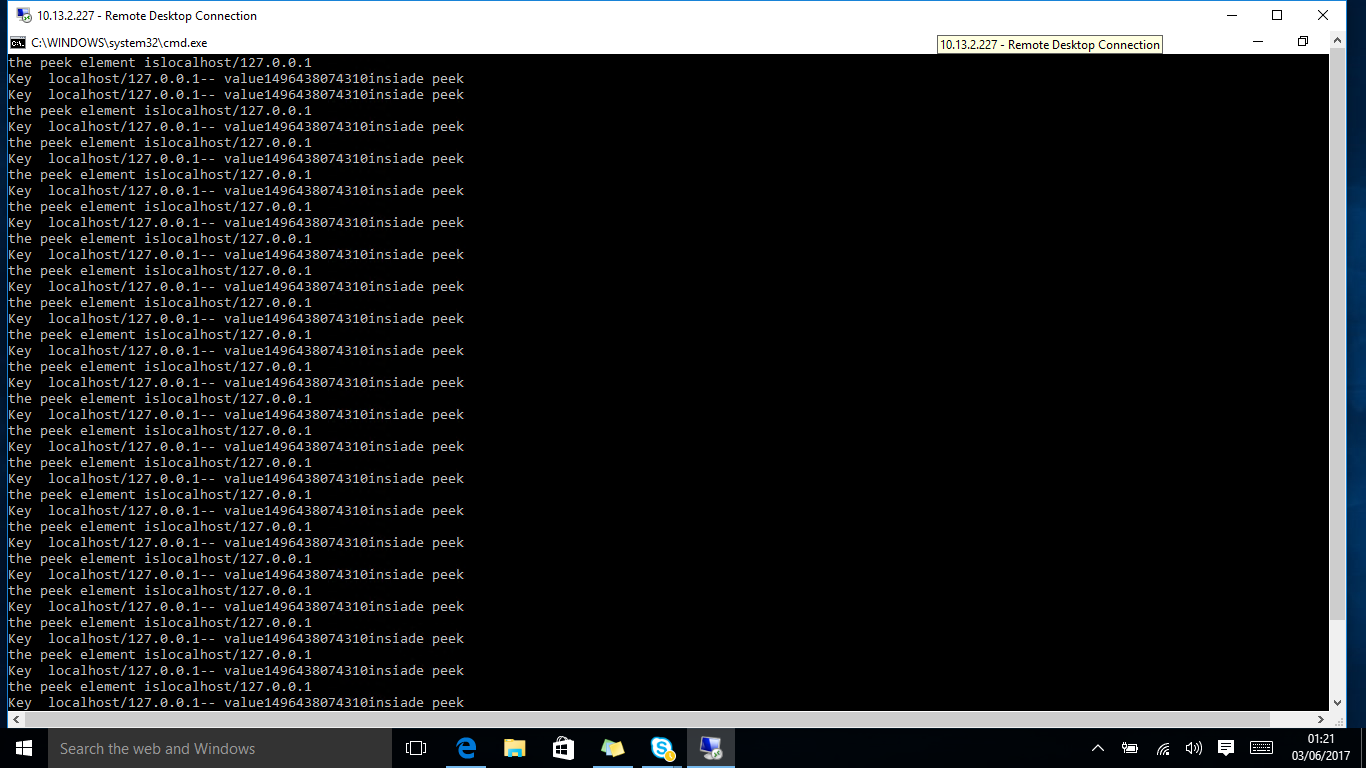
All the methods are tested at once by running multiple threads. The screenshots of each has been given.

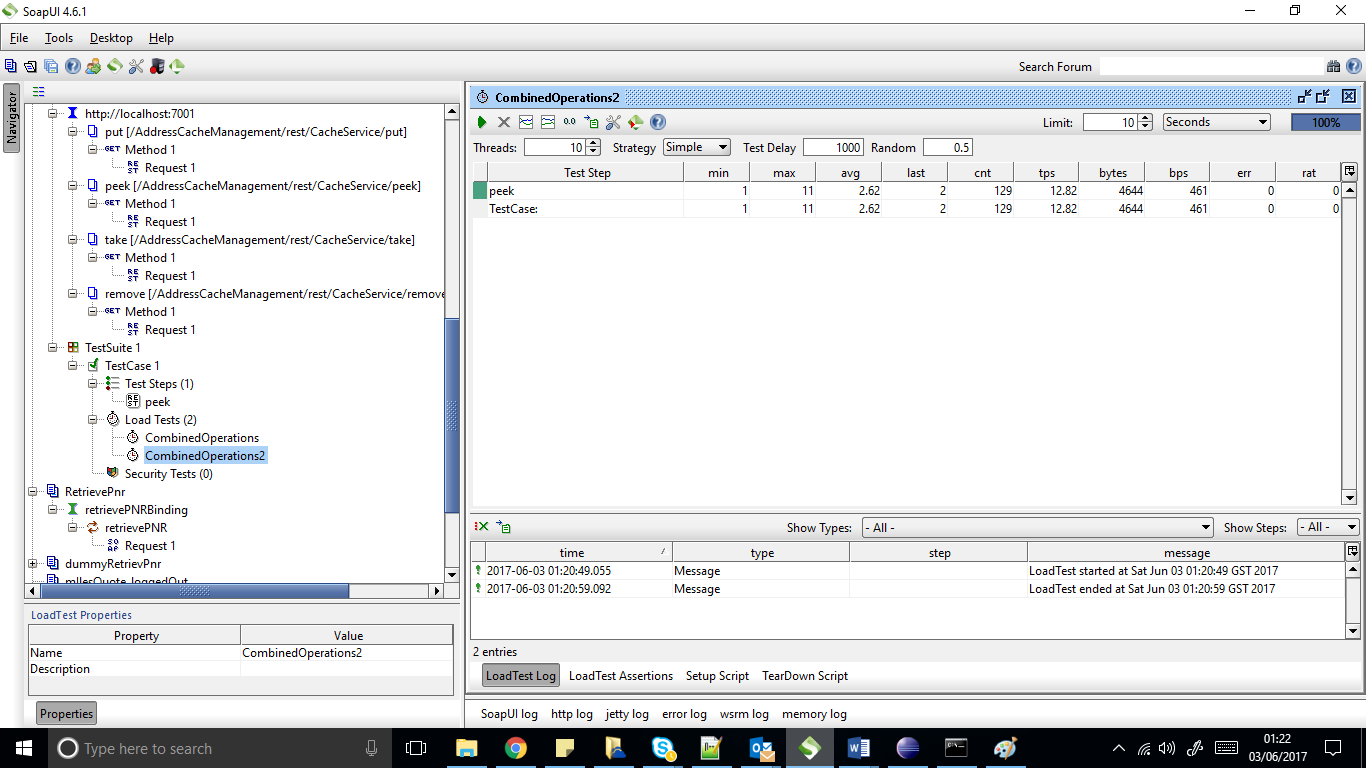
1. Multithreading with PUT operation (element is added once. All the other threads get already present in cache).It gets executed without any errors giving expected response.



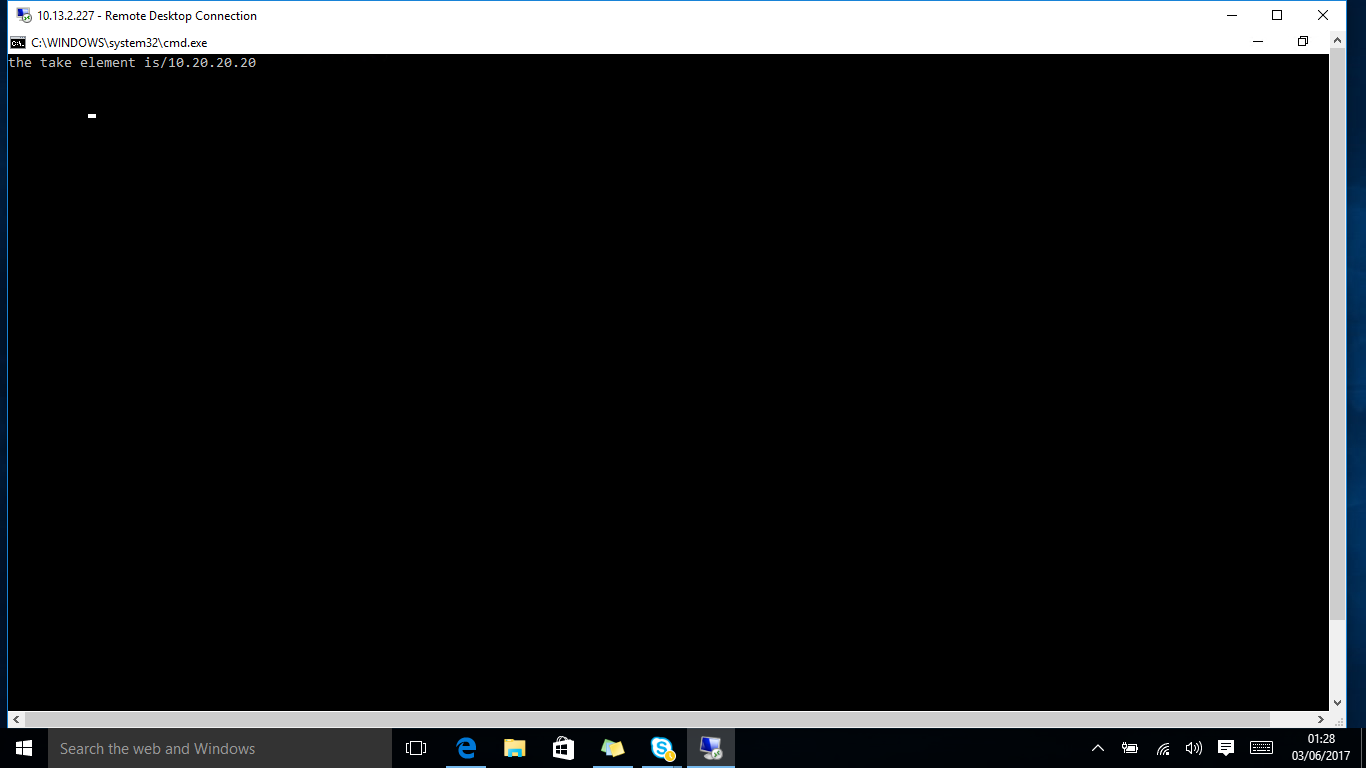


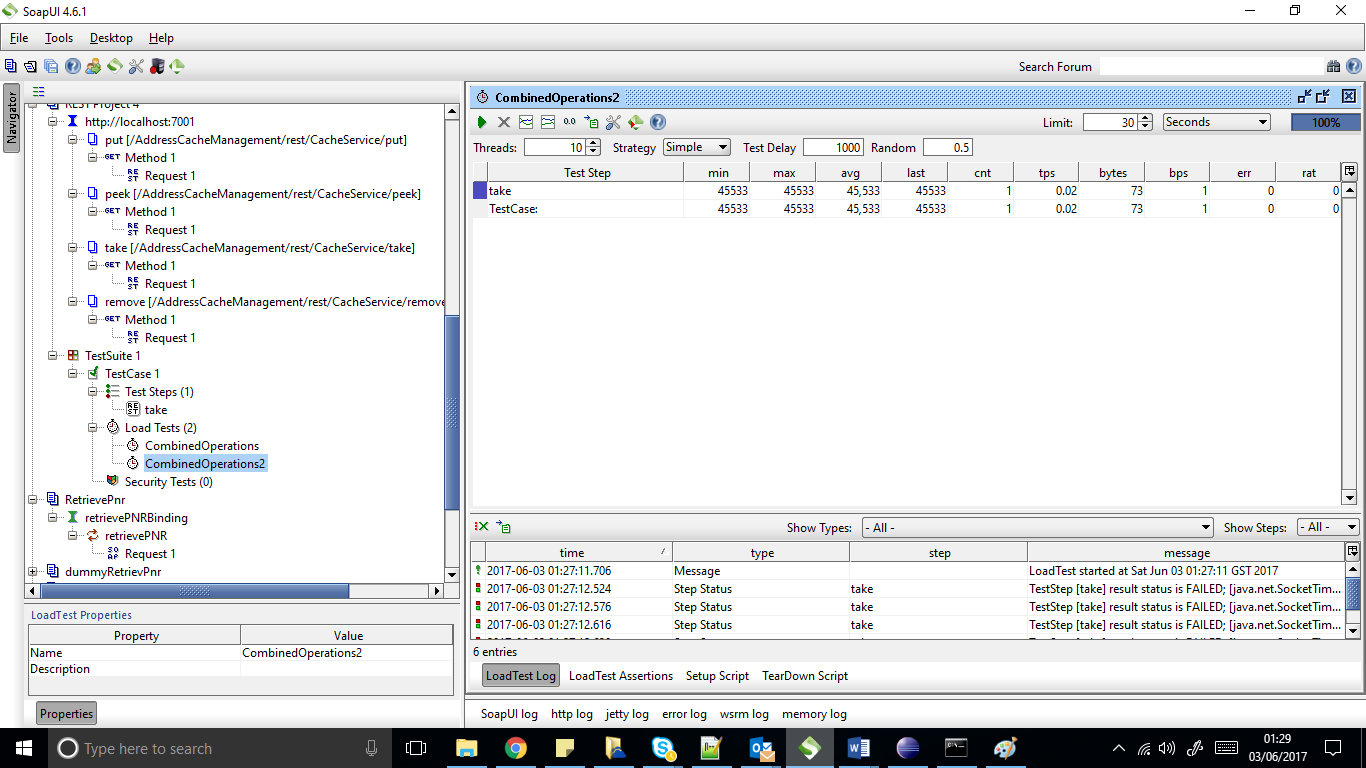
1. PEEK operation – All the threads get the same recent added element



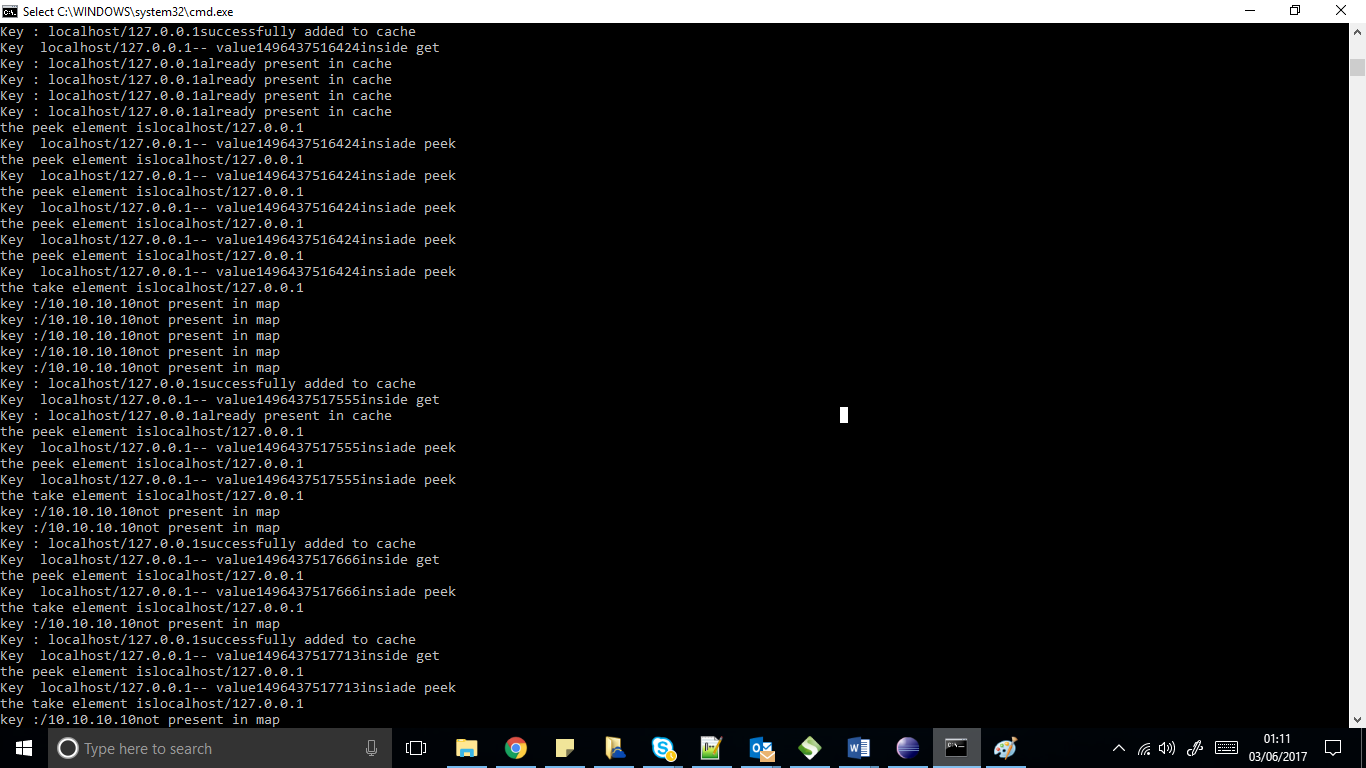


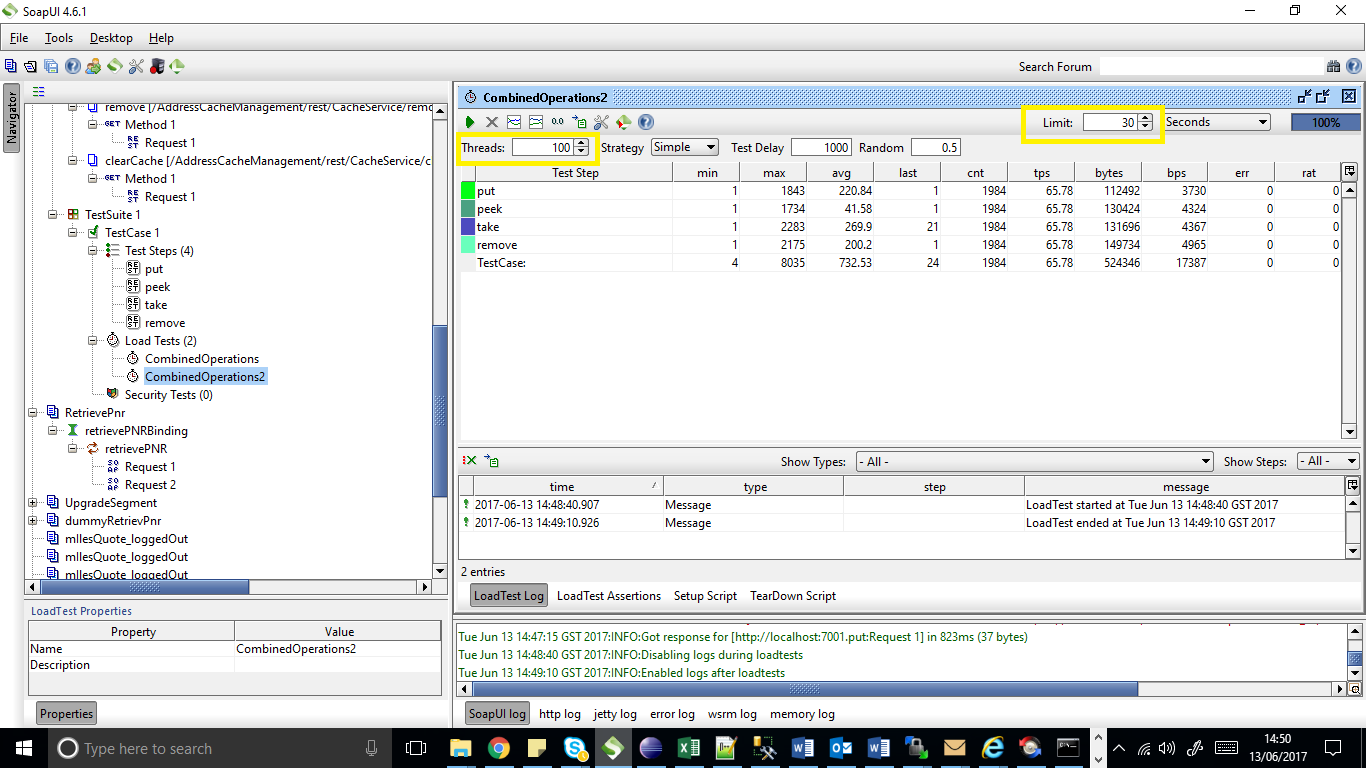
1. TAKE operation – the take operation has been implemented specially. The take operation will get the recently added element and remove it. Incase the cache is empty the thread will sleep for sometime and keep on checking the cache for it to be not null , the retry attempt is also kept configurable.





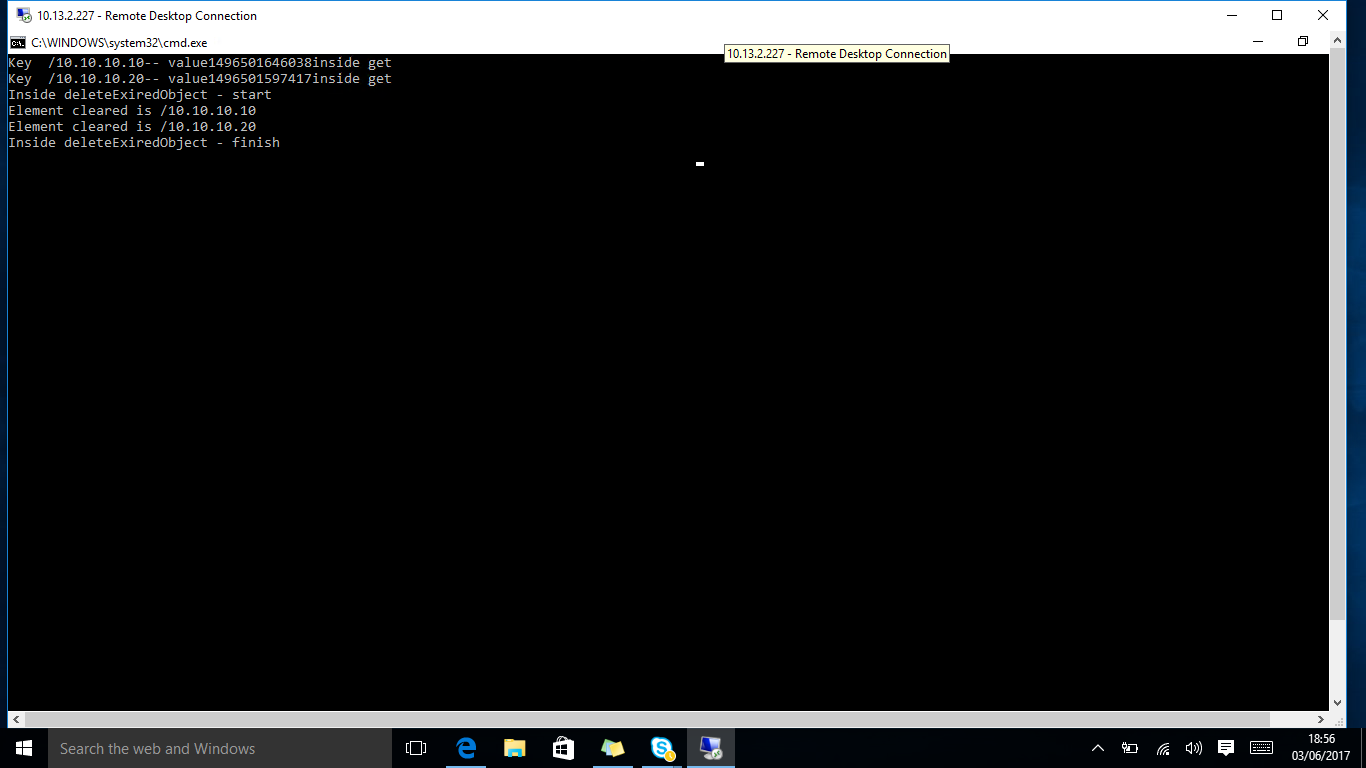
1. Combined operations as a part of multithreading operations. The user can specify the number of threads to be used.(I used 100 threads for testing in 30sec)





All threads have executed with zero errors giving expected response. I have done a thorough research before implementing this solution.

1. CLEAR CACHE – The clear cache runs every 2 mins and object expiry time has been kept as 3 mins. The scheduler checks every 2 mins for elements which have expired their age and clears them . The times are kept as constant in a constant file.



1. **DESIGN – SCALES THOUSANDS READ , WRITE PER SEC**

The Concurrent hash map has been implemented to improvise performance. The number of threads or the number of buckets in which a concurrent hash map can be divided can be configured. The usage of concurrent hash maps for caches has been considered the best as per the POC done. It has scaled well in enterprise applications catering to millions of users.

1. **SHOULD THE CODE BE OPTIMIZED FOR READS,WRITES OR MIXED**

The implementation using a concurrent hash map is already optimized in its own ways. But if it wouldn’t have been a map, we could have used copy on write array list or blocking queues for concurrency purposes , but these implementations are good only for read purpose, not for writing purpose. Since our application has writes as well, I preferred using map.

The optimization for performance can be done by using asynchronous thread activity using a thread pool executor in future which carries out each operation in a separate thread to further save timing constraints if any, since it is a very small application with not much of processing asynchronous thread mechanisms have not been used.

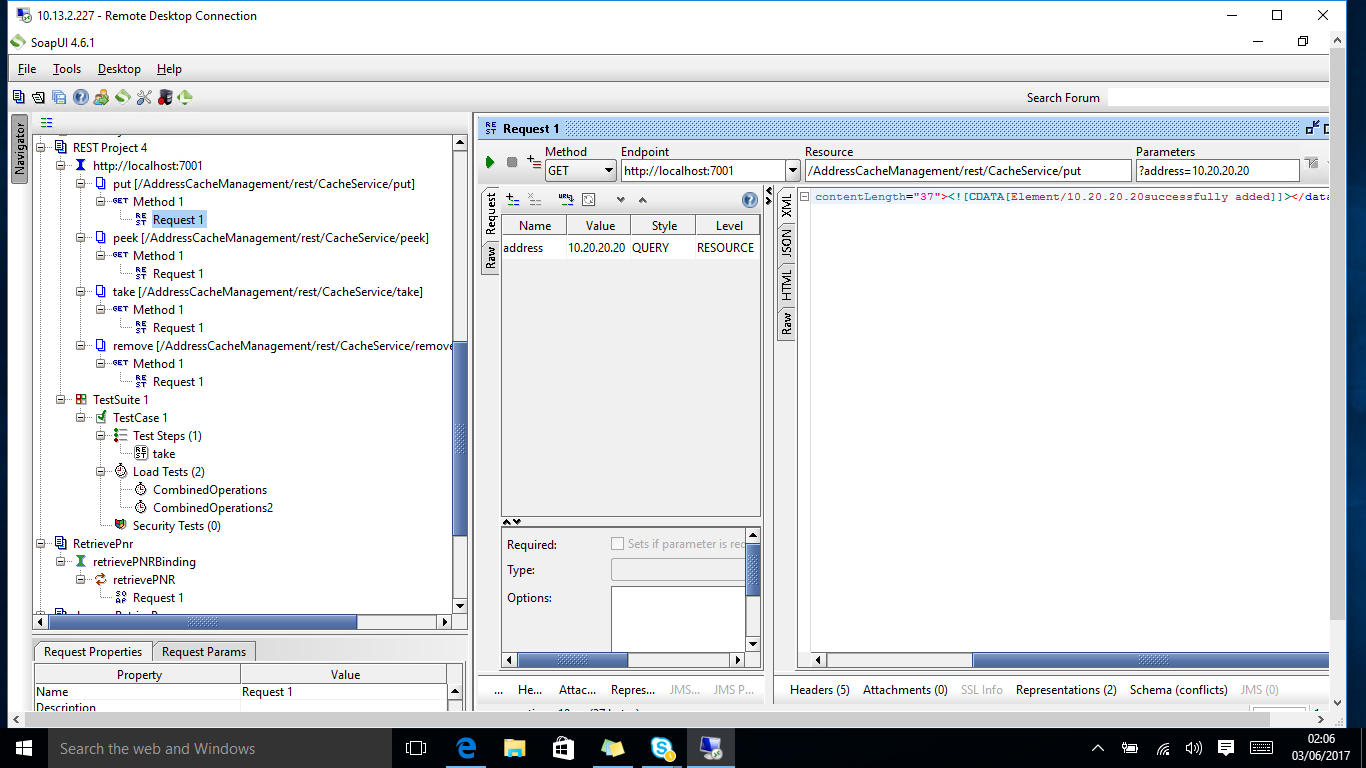
1. **TEST COVERAGE**

The testing has been done for each operation using SOAP UI. For each operation using threads along with all the operations at one go with multiple threads. The testing has been done successfully without any errors. I have used 500 threads a time to check if there are concurrency issues but it executed without any errors.

The JUNIT testing has also been implemented to check the correctness of the methods returning the expected values.

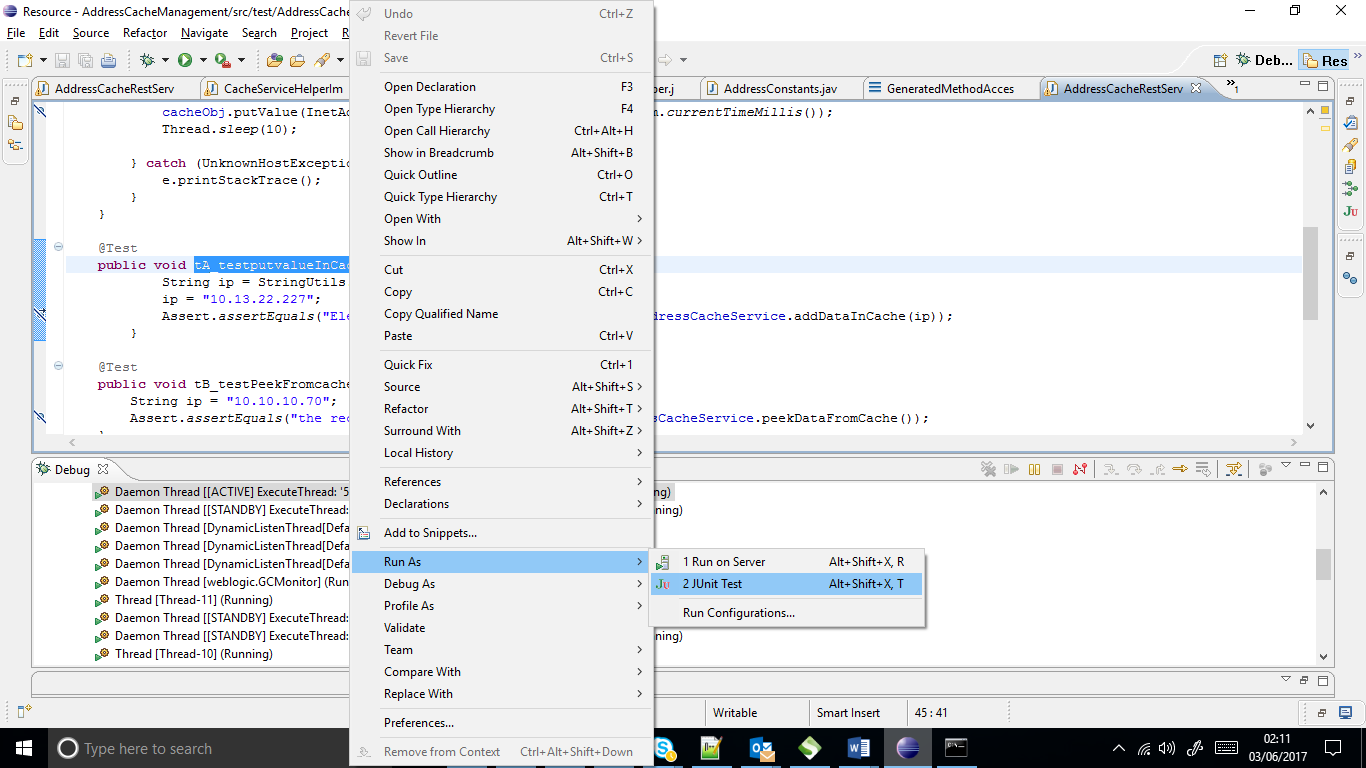
1. **USING THE API – BASIC CONSTRUCTION**
2. The project has all the necessary jar files required for running the project.
3. Step 1 – import the project in Eclipse or any other IDE.
4. If in case, the project shows error, please right click the project🡪 build path. Remove the jars from the build path and add external jars from the web-inf (lib)folder of the project.
5. Go to file 🡪 export as a war file (give the build directory). The build is ready
6. Place the build in your server domain. For me I have used weblogic as the server domain.
7. From SOAP UI, put the url’s mentioned in the earlier part of the document. Get the result as a string.

DEMO



JUNIT TEST CASES  
1) Open the test class – I have added a test class for positive test cases and the other for negative test case. Select a method for which you need to run the test case and right click and select run as junit test case.

NOTE – please run the cases as per the given sequence, since it is a string matcher and I have tried to match the strings using pre populated dummy entry in the cache.



PLEASE NOTE –

I have not used loggers to log the data, but I have printed the results in console.

So the operations can be tracked in the console.