ConcurrentHashMap in java

**Prerequisites:**Need of ConcurrentMap  
**ConcurrentHashMap** ConcurrentHashMap class is introduced in JDK 1.5, which implements ConcurrentMap as well as Serializable interface also. ConcureentHashMap is enhancement of HashMap as we know that while dealing with Threads in our application HashMap is not a good choice because performance wise HashMap is not upto the mark.

**Key points of ConcurrentHashMap:**

* The underlined data structure for ConcurrentHashMap is Hashtable.
* ConcurrentHashMap class is thread-safe i.e. multiple thread can operate on a single object without any complications.
* At a time any number of threads are applicable for read operation without locking the ConcurrentHashMap object which is not there in HashMap.
* In ConcurrentHashMap, the Object is divided into number of segments according to the concurrency level.
* Default concurrency-level of ConcurrentHashMap is 16.
* In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updation in object, thread must lock the particular segment in which thread want to operate.This type of locking mechanism is known as **Segment locking or bucket locking**.Hence at a time 16 updation operations can be performed by threads.
* null insertion is not possible in ConcurrentHashMap as key or value.

**Constructors of ConcurrentHashMap:**

1. **ConcurrentHashMap m=new ConcurrentHashMap();**:Creates a new, empty map with a default initial capacity (16), load factor (0.75) and concurrencyLevel (16).
2. **ConcurrentHashMap m=new ConcurrentHashMap(int initialCapacity);**:Creates a new, empty map with the specified initial capacity, and with default load factor (0.75) and concurrencyLevel (16).
3. **ConcurrentHashMap m=new ConcurrentHashMap(int initialCapacity, float loadFactor);**:  
   Creates a new, empty map with the specified initial capacity and load factor and with the default concurrencyLevel (16).
4. **ConcurrentHashMap m=new ConcurrentHashMap(int initialCapacity, float loadFactor, int concurrencyLevel);**:Creates a new, empty map with the specified initial capacity, load factor and concurrency level.
5. **ConcurrentHashMap m=new ConcurrentHashMap(Map m);**:Creates a new map with the same mappings as the given map.

|  |
| --- |
| // Java program to demonstrate working of ConcurrentHashMap  import java.util.concurrent.\*;  class ConcurrentHashMapDemo {      public static void main(String[] args)      {          ConcurrentHashMap m = new ConcurrentHashMap();          m.put(100, "Hello");          m.put(101, "Geeks");          m.put(102, "Geeks");            // Here we cant add Hello because 101 key          // is already present in ConcurrentHashMap object          m.putIfAbsent(101, "Hello");            // We can remove entry because 101 key          // is associated with For value          m.remove(101, "Geeks");            // Now we can add Hello          m.putIfAbsent(103, "Hello");            // We cant replace Hello with For          m.replace(101, "Hello", "For");          System.out.println(m);      }  } |

Output:

{100=Hello, 102=Geeks, 103=Hello}

**Methods in ConcurrentHashMap:**

1. **[clear​()](https://www.geeksforgeeks.org/java-concurrenthashmap-clear/):** This method removes all of the mappings from this map.
2. **[compute​(K key, BiFunction<K, V> remappingFunction)](https://www.geeksforgeeks.org/concurrenthashmap-compute-method-in-java-with-examples/):** This method attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
3. **[computeIfAbsent​(K key, Function<K, V> mappingFunction)](https://www.geeksforgeeks.org/concurrenthashmap-computeifabsent-method-in-java-with-examples/):** This method computes if the specified key is not already associated with a value, attempts to compute its value using the given mapping function and enters it into this map unless null.
4. **computeIfpresent​(K key, BiFunction<K,V> remappingFunction):** This method computes if the value for the specified key is present, attempts to compute a new mapping given the key and its current mapped value.
5. **[contains​(Object value)](https://www.geeksforgeeks.org/concurrenthashmap-contains-method-in-java-with-examples/):** This method tests if some key maps into the specified value in this table.
6. **[containsKey​(Object key)](https://www.geeksforgeeks.org/concurrenthashmap-containskey-method-in-java/):** This method tests if the specified object is a key in this table.
7. **[containsValue​(Object value](https://www.geeksforgeeks.org/concurrenthashmap-containsvalue-method-in-java/)):** This method returns true if this map maps one or more keys to the specified value.
8. **[elements​()](https://www.geeksforgeeks.org/concurrenthashmap-elements-method-in-java-with-examples/):** This method returns an enumeration of the values in this table.
9. **[entrySet​()](https://www.geeksforgeeks.org/concurrenthashmap-entryset-method-in-java-with-examples/):** This method returns a Set view of the mappings contained in this map.
10. **equals​(Object o):** This method Compares the specified object with this map for equality.
11. **forEach(long parallelismThreshold, BiConsumer<K, V> action):** This method performs the given action for each (key, value).
12. **forEach​(long parallelismThreshold, BiFunction<K,V> transformer, Consumer<U> action):** This method performs the given action for each non-null transformation of each (key, value).
13. **forEach​(BiConsumer<K, V> action):** This method performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
14. **forEachEntry​(long parallelismThreshold, Consumer<Map.Entry<K,V>> action):** This method performs the given action for each entry.
15. **forEachEntry​(long parallelismThreshold, Function<Map.Entry<K,V>> transformer, Consumer<U> action):** This method performs the given action for each non-null transformation of each entry.
16. **forEachKey​(long parallelismThreshold, Consumer<K> action):** This method performs the given action for each key.
17. **forEachKey​(long parallelismThreshold, Function<K> transformer, Consumer<U> action):** This method performs the given action for each non-null transformation of each key.
18. **forEachValue​(long parallelismThreshold, Consumer<V> action):** This method performs the given action for each value.
19. **forEachValue​(long parallelismThreshold, Function<V, U> transformer, Consumer<U> action):** This method performs the given action for each non-null transformation of each value.
20. **[get​(Object key)](https://www.geeksforgeeks.org/concurrenthashmap-get-method-in-java/):** This method returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
21. **getOrDefault​(Object key, V defaultValue):** This method returns the value to which the specified key is mapped, or the given default value if this map contains no mapping for the key.
22. **hashCode​():** This method returns the hash code value for this Map, i.e., the sum of, for each key-value pair in the map, key.hashCode() ^ value.hashCode().
23. **[isEmpty​()](https://www.geeksforgeeks.org/concurrenthashmap-isempty-method-in-java/):** This method returns true if this map contains no key-value mappings.
24. **[keys​()](https://www.geeksforgeeks.org/concurrenthashmap-keys-method-in-java-with-examples/):** This method returns an enumeration of the keys in this table.
25. **[keySet​()](https://www.geeksforgeeks.org/concurrenthashmap-keyset-method-in-java-with-examples/):** This method returns a Set view of the keys contained in this map.
26. **[keySet​(V mappedValue)](https://www.geeksforgeeks.org/concurrenthashmap-keyset-method-in-java-with-examples/):** This method returns a Set view of the keys in this map, using the given common mapped value for any additions (i.e., Collection.add(E) and Collection.addAll(Collection)).
27. **mappingCount​():** This method returns the number of mappings.
28. **merge​(K key, V value, BiFunction<K, V> remappingFunction):** This method If the specified key is not already associated with a (non-null) value, associates it with the given value.
29. **newKeySet​():** This method Creates a new Set backed by a ConcurrentHashMap from the given type to Boolean.TrUE.
30. **newKeySet​(int initialCapacity):** This method Creates a new Set backed by a ConcurrentHashMap from the given type to Boolean.TrUE.
31. **[put​(K key, V value)](https://www.geeksforgeeks.org/concurrenthashmap-put-method-in-java/):** This method Maps the specified key to the specified value in this table.
32. **[putAll​(Map<K, V> m)](https://www.geeksforgeeks.org/concurrenthashmap-putall-method-in-java/):** This method Copies all of the mappings from the specified map to this one.
33. **[putIfAbsent​(K key, V value)](https://www.geeksforgeeks.org/concurrenthashmap-putifabsent-method-in-java/):** This method If the specified key is not already associated with a value, associates it with the given value.
34. **reduce​(long parallelismThreshold, BiFunction<K, V> transformer, BiFunction<U, V> reducer):** This method returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, or null if none.
35. **reduceEntries​(long parallelismThreshold, BiFunction<Map.Entry<K,V>> reducer):** This method returns the result of accumulating all entries using the given reducer to combine values, or null if none.
36. **reduceEntries​(long parallelismThreshold, Function<Map.Entry<K,V>> transformer, BiFunction<U, V> reducer):** This method returns the result of accumulating the given transformation of all entries using the given reducer to combine values, or null if none.
37. **reduceEntriesToDouble​(long parallelismThreshold, ToDoubleFunction<Map.Entry<K,V>> transformer, double basis, DoubleBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.
38. **reduceEntriesToInt​(long parallelismThreshold, ToIntFunction<Map.Entry<K,V>> transformer, int basis, IntBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.
39. **reduceEntriesToLong​(long parallelismThreshold, ToLongFunction<Map.Entry<K,V>> transformer, long basis, LongBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.
40. **reduceKeys​(long parallelismThreshold, BiFunction<K> reducer):** This method returns the result of accumulating all keys using the given reducer to combine values, or null if none.
41. **reduceKeys​(long parallelismThreshold, Function<K> transformer, BiFunction<U> reducer):** This method returns the result of accumulating the given transformation of all keys using the given reducer to combine values, or null if none.
42. **reduceKeysToDouble​(long parallelismThreshold, ToDoubleFunction<K> transformer, double basis, DoubleBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.
43. **reduceKeysToInt​(long parallelismThreshold, ToIntFunction<K> transformer, int basis, IntBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.
44. **reduceKeysToLong​(long parallelismThreshold, ToLongFunction<K> transformer, long basis, LongBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.
45. **reduceToDouble​(long parallelismThreshold, ToDoubleBiFunction<K, V> transformer, double basis, DoubleBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.
46. **reduceToInt​(long parallelismThreshold, ToIntBiFunction<K, V> transformer, int basis, IntBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.
47. **reduceToLong​(long parallelismThreshold, ToLongBiFunction<K, V> transformer, long basis, LongBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.
48. **reduceValues​(long parallelismThreshold, BiFunction<V> reducer):** This method returns the result of accumulating all values using the given reducer to combine values, or null if none.
49. **reduceValues​(long parallelismThreshold, Function<V, U> transformer, BiFunction<U> reducer):** This method returns the result of accumulating the given transformation of all values using the given reducer to combine values, or null if none.
50. **reduceValuesToDouble​(long parallelismThreshold, ToDoubleFunction<V> transformer, double basis, DoubleBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.
51. **reduceValuesToInt​(long parallelismThreshold, ToIntFunction<V> transformer, int basis, IntBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.
52. **reduceValuesToLong​(long parallelismThreshold, ToLongFunction<V> transformer, long basis, LongBinaryOperator reducer):** This method returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.
53. **[remove​(Object key)](https://www.geeksforgeeks.org/concurrenthashmap-remove-method-in-java/):** This method removes the key (and its corresponding value) from this map.
54. **[remove​(Object key, Object value)](https://www.geeksforgeeks.org/concurrenthashmap-remove-method-in-java/):** This method removes the entry for a key only if currently mapped to a given value.
55. **replace​(K key, V value):** This method replaces the entry for a key only if currently mapped to some value.
56. **replace​(K key, V oldValue, V newValue):** This method replaces the entry for a key only if currently mapped to a given value.
57. **replaceAll​(BiFunction<K, V> function):** This method replaces each entry’s value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
58. **search​(long parallelismThreshold, BiFunction<K, V, U> searchFunction):** This method returns a non-null result from applying the given search function on each (key, value), or null if none.
59. **searchEntries​(long parallelismThreshold, Function<Map.Entry<K,V>, U> searchFunction):** This method returns a non-null result from applying the given search function on each entry, or null if none.
60. **searchKeys​(long parallelismThreshold, Function<K, U> searchFunction):** This method returns a non-null result from applying the given search function on each key, or null if none.
61. **searchValues​(long parallelismThreshold, Function<V, U> searchFunction):** This method returns a non-null result from applying the given search function on each value, or null if none.
62. **[size​()](https://www.geeksforgeeks.org/concurrenthashmap-size-method-in-java/):** This method returns the number of key-value mappings in this map.
63. **toString​():** This method returns a string representation of this map.
64. **[values​()](https://www.geeksforgeeks.org/concurrenthashmap-values-method-in-java-with-examples/):** This method returns a Collection view of the values contained in this map.