Map<K,V> Interface

In previous post we saw some introduction on Map<K,V> interface. Now we will take a deep dive into Map<K,V> interface.

Map<K, V> means an object that maps keys to values. A map cannot contain duplicate keys. And each key can map to one and only one value.

Map interface models the mathematical function abstraction.

HashMap<K, V>, LinkedHashMap<K, V>, TreeMap<K, V>, ConcurrentHashMap<K, V> are the most popular concrete implementations of Map<K, V> interface.

**int size();** - size() method returns the number of Key-Value mappings in the map. If map contains more than Integer.MAX\_VALUE then it returns Integer.MAX\_VALUE.

**boolean isEmpty();** - returns true if map is empty or size is 0 or there are no Key-Value mapping in map.

**boolean** **containsKey(Object key);** - Returns true of map contains the key specified in parameter. Returns true if map contains mapping key k for key such that (key==null ? k==null : key.equals(k)). There can be only one such mapping.

**boolean containsValue(Object value);** - Returns true if map contains one or more keys mapped to same value. Yes, different keys can have same values. It’s legal to do that.

**V get(Object key);** - returns the value from map to which the key is associated. If no key is found then null is returned. It is permitted to have more than one null values in map. So get(Object key) method can return null as value value. So use containsKey(Object key) to check for key and then search for value to distinguish between null value or no value in map.

**V put(K key, V value);** - This method is used to insert the pair K key and V value into the map. If map does contains the key K then it replaces the previous value with the new value specified in parameter. This method returns the previous value.

**V remove(Object key);** - Removes the mapping for key from the map if it is present in map. Returns the value the was previously associated with the key. If this map permits the null values then null is returned. Also, if there is not key present in map then also null is returned. So make sure to call containsKey(Object key) first then call remove(Object key).

**void putAll(Map<? extends K, ? extends V> m);** - This method is used to copy all the mappings from specified map to this map.

**void clear();** - This method is used to remove all the mappings from this map. One the call returns the map will be empty or size = 0 or there won’t be any key to value associations.

Now we will look at the 3 different methods of how to view a map.

**Set<K> keySet();** - returns the Set<K> of keys. Set is backed by Map so changes of Map are reflected in Set too and vice versa. Returns set view of keys contained in map.

**Collection<V> values();** - returns collection view of values.

**Set<Map.Entry<K, V>> entrySet();** - returns set of entry of map i.e. returns set of key-value associations. Set is backed by map so changes of map are reflected in set and vice versa.

In next post we will see the static interface Entry<K, V> declared in Map<K, V> interface.