Methods of Map<K,V> interface :

* The following are the commonly used methods inside Map<K,V> interface.

1) V put(Object key, Object value) :- To insert one entry in the Map collection. It will return the value of old Object key, if the key is already available(Duplicate key), If key is not available (new key) then it will return null.

2)default V putIfAbsent(Object key, Object value) :- It will insert an entry, if and only if, key is not available , if the key is already available then it will not insert the Entry to the Map Collection

3) V get(Object key) :- It will return corresponding key value, if the key is not available then it will return null.

4) Object getOrDefault(Object key, Object defaultValue) :- To avoid null value this method has been introduced from JDK 1.8V, here we can pass some defaultValue to avoid the null value.

5) boolean containsKey(Object key) :- To Search a particular key

6) boolean containsValue(Object value) :- To Search a particular value

7) int size() :- To count the number of Entries.

8) remove(Object key) :- One complete entry will be removed.

9) void clear() :- Used to clear the Map

10) boolean isEmpty() :- To verify Map is empty or not?

11) void putAll(Map m) :- Merging of two Map collection

12) default void forEach(BiConsumer<T,U> cons) :- Used to fetch key and value from the Map. It is added inside Map<K,V> interface from
JDK 1.8V

13) V computeIfAbsent(K key, Function<T,R> mapper) : It computes a value for a given key if it is not already present in the map, and then inserts that computed value into the map.

```
Map<String, String> map = new HashMap<>();
  map.computeIfAbsent("language", key -> "Java");
  map.computeIfAbsent("language", key -> "Python");
  System.out.println(map);
```

14) V computeIfPresent(K key, BiFunction<T,U,R> mapper) : It updates the value for a key only if that key is already available in the map.

```
Map<String, Integer> map = new HashMap<>();
map.put("apple", 10);
map.put("banana", 5);

map.computeIfPresent("apple", (key, val) -> val + 5);
map.computeIfPresent("orange", (key, val) -> val + 5);
System.out.println(map);
```

Methods of Map interface to convert the Map into Collection :

We have Map interafce methods through which we can convert Map interface into Collection interface which is known as collection views method.

1) public Set<Object> keySet() : It will retrieve all the keys.

2) public Collection values() : It will retrieve all the values.

3) public Set<Map.Entry<K,V>> entrySet : It will retrieve key and value both in a single object.
   a) getKey()
   b) getValue()

HashMap<K,V> [Un-Sorted, UnOrdered, No Duplicate key]

public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V> , Cloneable, Serializable

* It is an implemented class of Map<K,V> interface available from JDK 1.2V
* It does not accept duplicate key but can accept duplicate values.
* It can accept homogeneous and hetrogeneous keys and values.
* It can accept only null key and multiple null values.
* It is an Un-sorted and Un-ordered map.
* It uses Hashtable data structure.
* The entry will be inserted based on the hashCode() of the key object.
* Default capacity is 16 so, initially 16 buckets will be created.
* Methods are not synchronized.
* Iterator is Fail Fast Iterator.
* It is mainly used to fetch the entry from the Map collection.
* It provides constant time performance for insertion, deletion and searching operation.

It contains 4 types of constructor

1) HashMap hm1 = new HashMap();
   It will create the HashMap Object with default capacity is 16. The default load fator or Fill Ratio is 0.75 (75% of HashMap is filled up then new HashMap Object will be created having double capacity)

2) HashMap hm2 = new HashMap(int initialCapacity);
   will create the HashMap object with user specified capacity

3) HashMap hm3 = new HashMap(int initialCapacity, float loadFactor);
   we can specify our own initialCapacity and loadFactor(by default load factor is 0.75)

4)HashMap hm4 = new HashMap(Map<K,V> m);
   Interconversion of Map Collection.