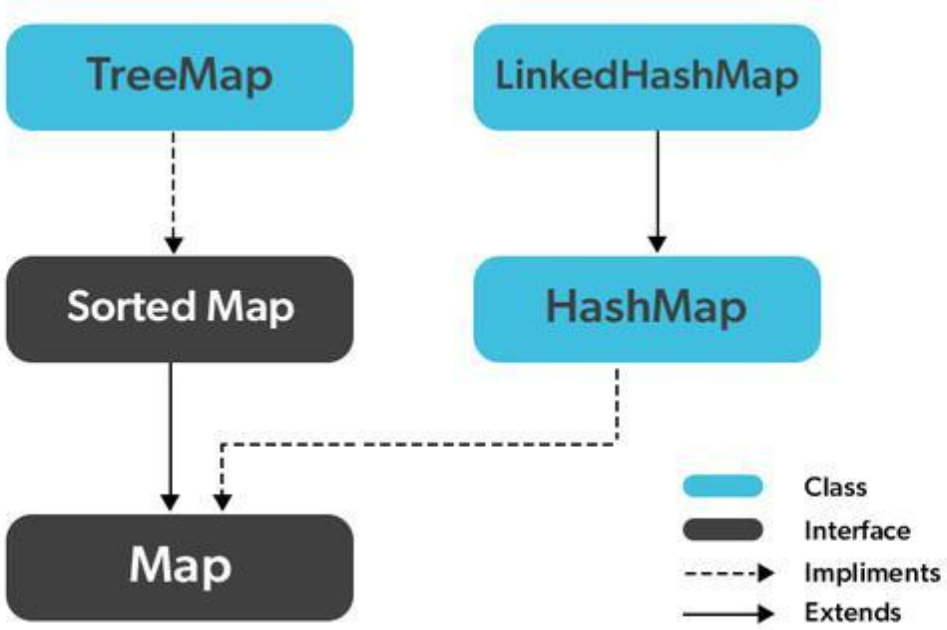
Map Interface - Java Collections

A Map stores data in key and value association. Both key and values are objects. The key must be unique but the values can be duplicate. Although Maps are a part of Collection Framework, they cannot actually be called as collections because of some of the properties that they possess. However, we can obtain a collection-view of maps.



It provides various classes: HashMap, TreeMap, LinkedHashMap for map implementation. All these classes implement Map interface to provide Map properties to the collection.

- **Map Interface and its Subinterface**

Interface	**Description**
Map	Maps unique key to value.
Map.Entry	Describe an element in key and value pair in a map. Entry is sub interface of Map.
SortedMap	Extends Map so that keys are maintained in an ascending order.

- **Map Interface Methods**

Method	Description
clear()	This method is used in Java Map Interface to clear and remove all of the elements or mappings from a specified Map collection.
containsKey(Object)	This method is used in Map Interface in Java to check whether a particular key is being mapped into the Map or not. It takes the key element as a parameter and returns True if that element is mapped in the map.
containsValue(Object)	This method is used in Map Interface to check whether a particular value is being mapped by a single or more than one key in the Map. It takes the value as a parameter and returns True if that value is mapped by any of the keys in the map.
entrySet()	This method is used in Map Interface in Java to create a set out of the same elements contained in the map. It basically returns a set view of the map or we can create a new set and store the map elements into them.
equals(Object)	This method is used in Java Map Interface to check for equality between two maps. It verifies whether the elements of one map passed as a parameter is equal to the elements of this map or not.
get(Object)	This method is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key.
hashCode()	This method is used in Map Interface to generate a hashCode for the given map containing keys and values.
isEmpty()	This method is used to check if a map is having any entry for key and value pairs. If no mapping exists, then this returns true.

Method	Description
keySet()	This method is used in Map Interface to return a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa.
put(Object, Object)	This method is used in Java Map Interface to associate the specified value with the specified key in this map.
putAll(Map)	This method is used in Map Interface in Java to copy all of the mappings from the specified map to this map.
remove(Object)	This method is used in Map Interface to remove the mapping for a key from this map if it is present in the map.
size()	This method is used to return the number of key/value pairs available in the map.
values()	This method is used in Java Map Interface to create a collection out of the values of the map. It basically returns a Collection view of the values in the HashMap.
getOrDefault(Object key, V defaultValue)	Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associate it with the given non-null value.
putIfAbsent(K key, V value)	If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current associate value.

- **HashMap class**

HashMap class extends AbstractMap and implements Map interface.
It uses a hashtable to store the map. This allows the execution time of get() and put() to remain same.
HashMap does not maintain order of its element.

HashMap has four constructor:

HashMap()

HashMap(Map< ? extends k, ? extends V> m)

HashMap(int capacity)

HashMap(int capacity, float fillratio)

HashMap Example

Lets take an example to create hashmap and store values in key and value pair. Notice to insert elements, we used put() method because map uses put to insert element, not add() method that we used in list interface.

```
import java.util.*;
class Demo{
    public static void main(String args[]){
        Map< String,Integer> hm = new HashMap< String,Integer>();
        hm.put("aniket",100);
        hm.put("bablu",200);
        hm.put("lokesh",900);
        hm.put("abcxyz",007);
        for(Map.Entry<String,Integer> P  : hm.entrySet()) {
            System.out.print(P.getKey()+":");
            System.out.println(P.getValue());
        }
    }
}
```

LinkedHashMap Example:

```
import java.util.*;
class Demo{
```

```
  public static void main(String args[]){
    Map< String,Integer> hm = new LinkedHashMap< String,Integer>();
    hm.put("nikhil",700);
    hm.put("shweta",789);
    hm.put("lokesh",900);
    hm.put("abcxyz",007);
    for(Map.Entry<String,Integer> P  : hm.entrySet()){
      System.out.println(P.getKey()+ " : " +P.getValue());
    }
  }
}
```

TreeMap Example:

```
import java.util.*;
class Demo{
  public static void main(String args[]){
    Map< String,Integer> hm = new TreeMap< String,Integer>();
    hm.put("aniket",100);
    hm.put("bablu",200);
    hm.put("cintu",300);
    hm.put("dany",400);
    for(Map.Entry<String,Integer> P  : hm.entrySet()){
      System.out.println(P.getKey()+ " : " +P.getValue());
    }
  }
}
```

How to iterate any Map in Java

There are generally five ways of iterating over a Map in Java.

1. Iterating over Map.entrySet() using For-Each loop.
2. Iterating over keys or values using keySet() and values() methods.
3. Iterating using iterators over Map.Entry<K, V>.
4. Using forEach(action) method.
5. Iterating over keys and searching for values (inefficient).

1. Iterating over Map.entrySet() using For-Each loop.

Map.entrySet() method returns a collection-view(Set<Map.Entry<K, V>>) of the mappings contained in this map. So we can iterate over key-value pair using getKey() and getValue() methods of Map.Entry<K, V>.

Example:

```
import java.util.*;
class Main {
   public static void main(String[] arg) {
      Map<String,String> hm = new HashMap<String,String>();

      hm.put("ABC", "1");
      hm.put("DEF", "2");
      hm.put("GHI", "3");
      hm.put("JKL", "4");

      for(Map.Entry<String, String> temp : hm.entrySet()) {
          System.out.println("K= " + temp.getKey() +  ", V= " + temp.getValue());
      }
   }
}
```

2. Iterating over keys or values using keySet() and values() methods.

Map.keySet() method returns a Set view of the keys contained in this map and Map.values() method returns a collection-view of the values contained in this map. So If you need only keys or values from the map, you can iterate over keySet or values using for-each loops.

Example:

```
import java.util.*;
class Main {
   public static void main(String[] arg) {
      Map<String,Integer> hm = new HashMap<String,Integer>();

      hm.put("ABC", 1);
      hm.put("DEF", 2);
      hm.put("GHI", 3);
      hm.put("JKL", 4);

      for(String str : hm.keySet()) {
         System.out.println("Key = " + str);
      }
      for(Integer i : hm.values()) {
         System.out.println("Value = " + i);
      }
   }
}
```

3. Iterating using iterators over Map.Entry<K, V>.

This method is somewhat similar to first one. In first method we use for-each loop over Map.Entry<K, V>, but here we use iterators. Using iterators over Map.Entry<K, V> has it's own advantage,i.e. we can remove entries from the map during iteration by calling iterator.remove() method.

Example:

```
import java.util.*;
class Main {
   public static void main(String[] arg) {
      Map<String,String> hm = new HashMap<String,String>();

      hm.put("ABC", "1");
      hm.put("DEF", "2");
      hm.put("GHI", "3");
      hm.put("JKL", "4");
```

```
        // using iterators
        Iterator<Map.Entry<String, String>> itr = hm.entrySet().iterator();
        while(itr.hasNext()) {
            Map.Entry<String, String> temp = itr.next();
            System.out.println("K= " + temp.getKey() +  ", V= " + temp.getValue());
        }
    }
}
```

4. Iterating Map Using forEach(action) method :

In Java 8, you can iterate a map using Map.forEach(action) method and using lambda expression. This technique is clean and fast.

Example:

```
import java.util.*;
class Main {
    public static void main(String[] arg) {
        Map<String,Integer> hm = new HashMap<String,Integer>();

        hm.put("ABC", 1);
        hm.put("DEF", 2);
        hm.put("GHI", 3);
        hm.put("JKL", 4);

        // forEach(action) method to iterate map
        hm.forEach((k,v) -> System.out.println("Key = "+ k +", Value = " + v));
    }
}
```

5. Iterating over keys and searching for values (inefficient).

Here first we loop over keys(using Map.keySet() method) and then search for value(using Map.get(key) method) for each key.This method is not used in practice as it is pretty slow and inefficient as getting values by a key might be time-consuming.

Example:

```
import java.util.*;
class Main {
   public static void main(String[] arg) {
      Map<String,Integer> hm = new HashMap<String,Integer>();

      hm.put("ABC", 1);
      hm.put("DEF", 2);
      hm.put("GHI", 3);
      hm.put("JKL", 4);

      for (String str : hm.keySet()){
         Integer i = hm.get(str);
         System.out.println("Key = "+ str +", Value = " + i);
      }
   }
}
```

Programs on Maps Interface and HashMap Class :

1. Adding Elements

```
import java.util.*;
class Demo{
  public static void main(String args[]){
    Map< String,Integer> hm1 = new HashMap< String,Integer>();
    Map< String,Integer> hm2 = new HashMap< String,Integer>();
    hm1.put("aniket",100);
    hm1.put("bablu",200);
    hm1.put("cintu",300);
    hm2.put("dany",400);
    hm2.put("nikhil",700);
    hm2.put("shweta",789);
    System.out.println(hm1);
    System.out.println(hm2);
  }
}
```

2. Changing Elements

```
import java.util.*;
class Demo{
  public static void main(String args[]){
    Map< Integer,String> hm1 = new HashMap< Integer,String>();
    hm1.put(100,"aniket");
    hm1.put(200,"bablu");
    hm1.put(300,"cintu");
    hm1.put(400,"dany");
    System.out.println("Old Map: "+hm1);

    hm1.put(300, "Updated");
    System.out.println("Updated Map: "+hm1);
  }
}
```

3. Removing Element

```
import java.util.*;
class Demo{
  public static void main(String args[]){
    Map< Integer,String> hm1 = new HashMap< Integer,String>();
    hm1.put(100,"aniket");
    hm1.put(200,"bablu");
    hm1.put(300,"cintu");
    hm1.put(400,"dany");
    System.out.println("Old Map: "+hm1);

    hm1.remove(200);
    System.out.println("Updated Map: "+hm1);
  }
}
```

4. Occurrence of Numbers using HashMap

```
import java.util.*;
class Main {
  static void frequency(int arr[]){
    HashMap<Integer, Integer> hm = new HashMap<>();
    for (int i : arr) {
      if (hm.containsKey(i)) {
        hm.put(i, hm.get(i) + 1);
      }
      else {
        hm.put(i, 1);
      }
    }

    for (Map.Entry entry : hm.entrySet()) {
      System.out.println(entry.getKey() + " " + entry.getValue());
    }
  }

  public static void main(String[] args){
    int arr[] = {1,2,1,2,3,4,1,2,4,7,3,4,5,6,};
    frequency(arr);
  }
}
```