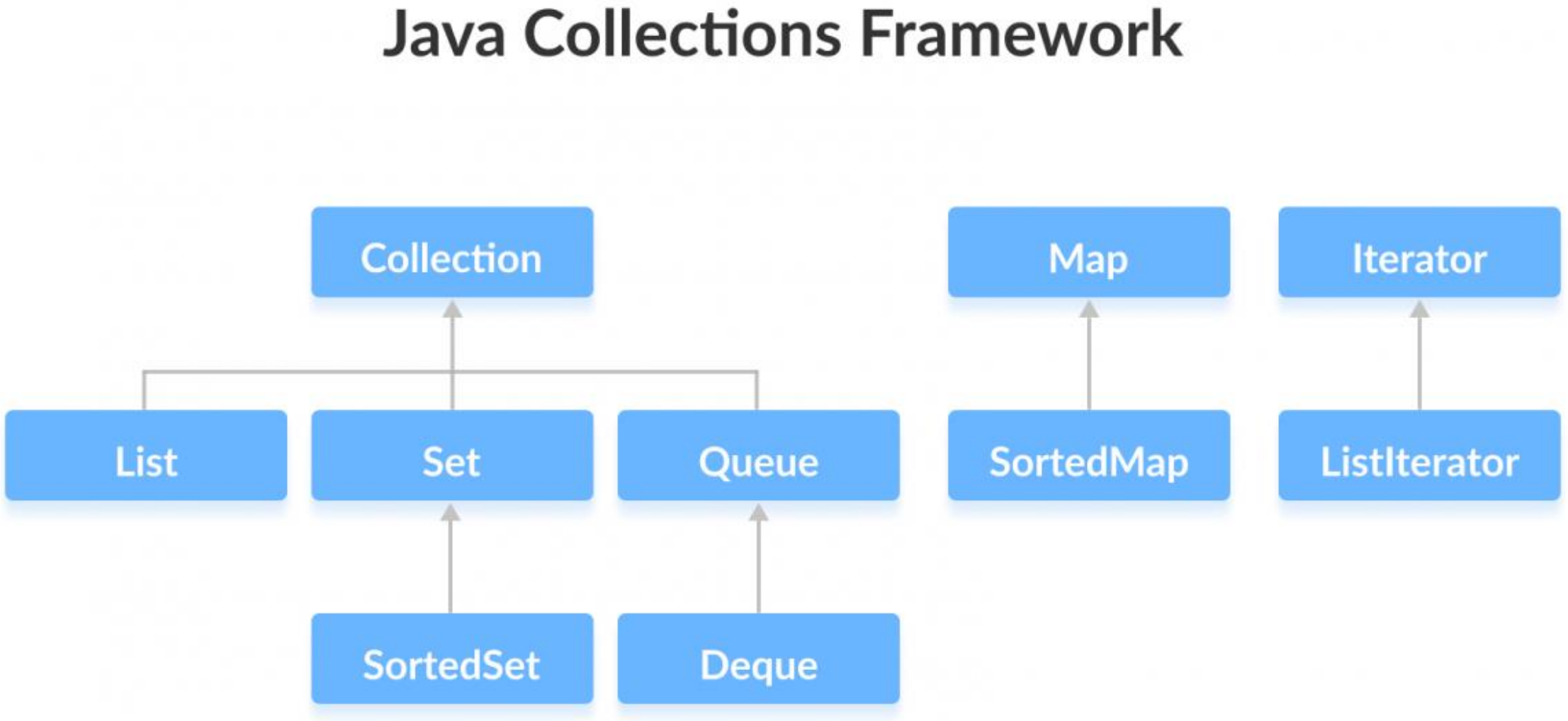
AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

19CSE204
Object Oriented Paradigm
2-0-3-3

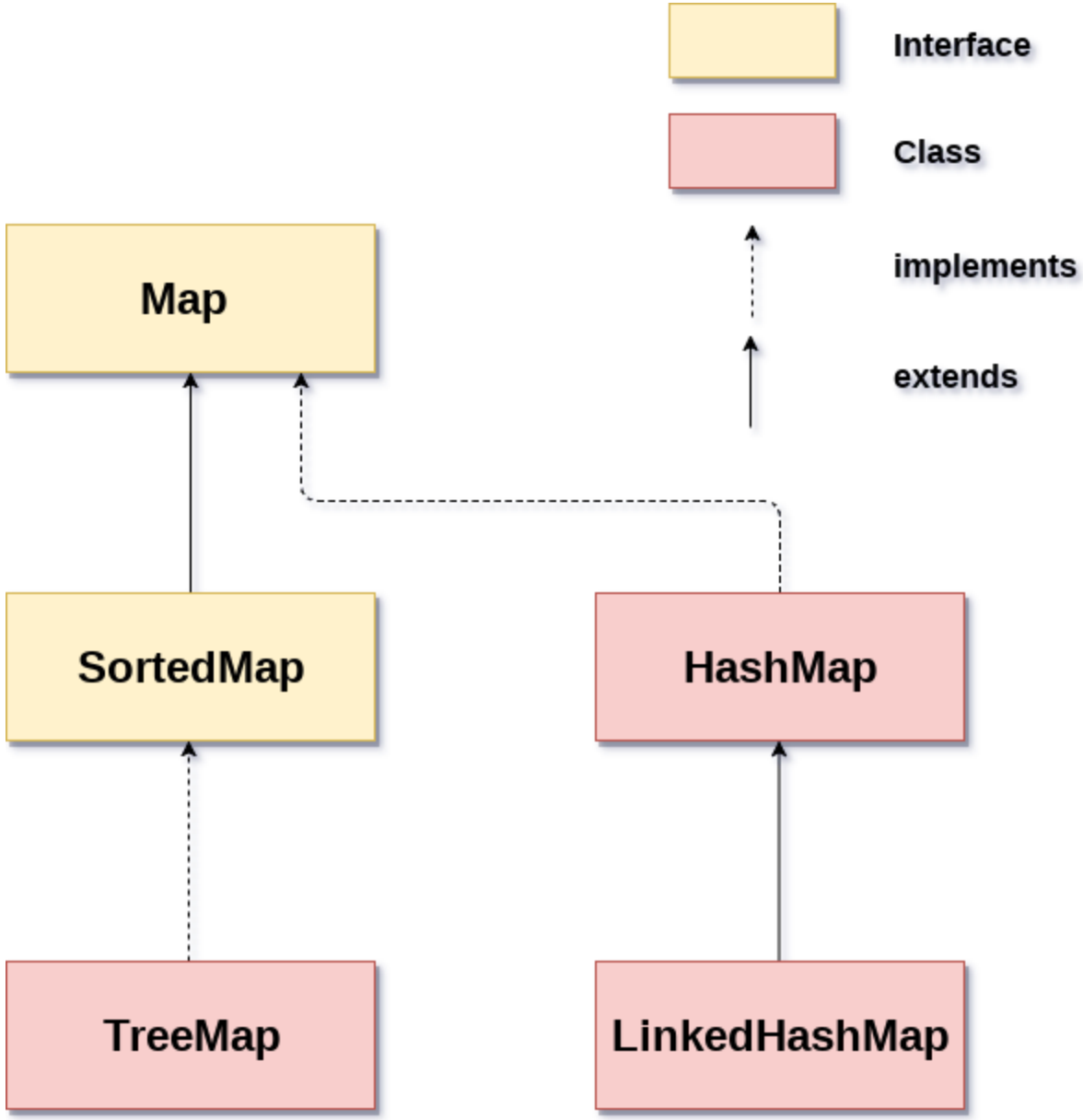
Java Map Interface

Recap


Java Collections Framework
Collection
Map
Iterator
List
Set
Queue
SortedMap
ListIterator
SortedSet
Deque

Java Map Interface

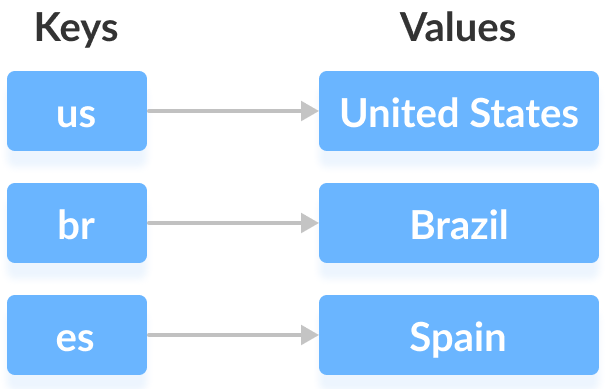
Java Map Interface



Java Map

- In Java, elements of Map are stored in key/value pairs. Keys are unique values associated with individual Values.

- A map cannot contain duplicate keys. And, each key is associated with a single value.



In the above diagram, we have values: United States, Brazil, and Spain. And we have corresponding keys: us, br, and es

How to use Map?

- In Java, we must import the java.util.Map package in order to use Map. Once we import the package, here's how we can create a map.

```
// Map implementation using HashMap
Map<Key, Value> numbers = new HashMap<>();
```

- In the above code, we have created a Map named numbers. We have used the HashMap class to implement the Map interface.

- Here,

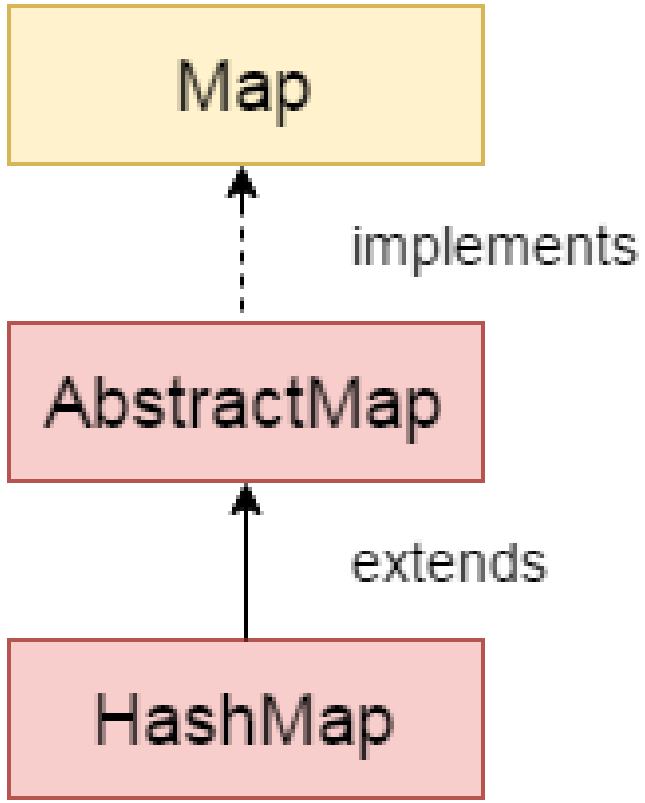
- Key - a unique identifier used to associate each element (value) in a map
- Value - elements associated by keys in a map

Map Methods (The Map interface includes all the methods of the Collection interface.In addition it has following methods too)

- **put(K, V)** - Inserts the association of a key K and a value V into the map. If the key is already present, the new value replaces the old value.
- **putAll()** - Inserts all the entries from the specified map to this map.
- **putIfAbsent(K, V)** - Inserts the association if the key K is not already associated with the value V.
- **get(K)** - Returns the value associated with the specified key K. If the key is not found, it returns null.
- **getOrDefault(K, defaultValue)** - Returns the value associated with the specified key K. If the key is not found, it returns the defaultValue.
- **containsKey(K)** - Checks if the specified key K is present in the map or not.
- **containsValue(V)** - Checks if the specified value V is present in the map or not.
- **replace(K, V)** - Replace the value of the key K with the new specified value V.
- **replace(K, oldValue, newValue)** - Replaces the value of the key K with the new value newValue only if the key K is associated with the value oldValue.
- **remove(K)** - Removes the entry from the map represented by the key K.
- **remove(K, V)** - Removes the entry from the map that has key K associated with value V.
- **keySet()** - Returns a set of all the keys present in a map.
- **values()** - Returns a set of all the values present in a map.

Java HashMap class

- HashMap contains value based on the key.
- It implements map interface and extends Abstract Map
- It contain only unique elements
- It may have one null key and multiple null values
- It maintains no order
- Difference between HashSet and HashMap
  - HashSet has only values whereas HasMap contains entry (key and value)



Implementing HashMap Class

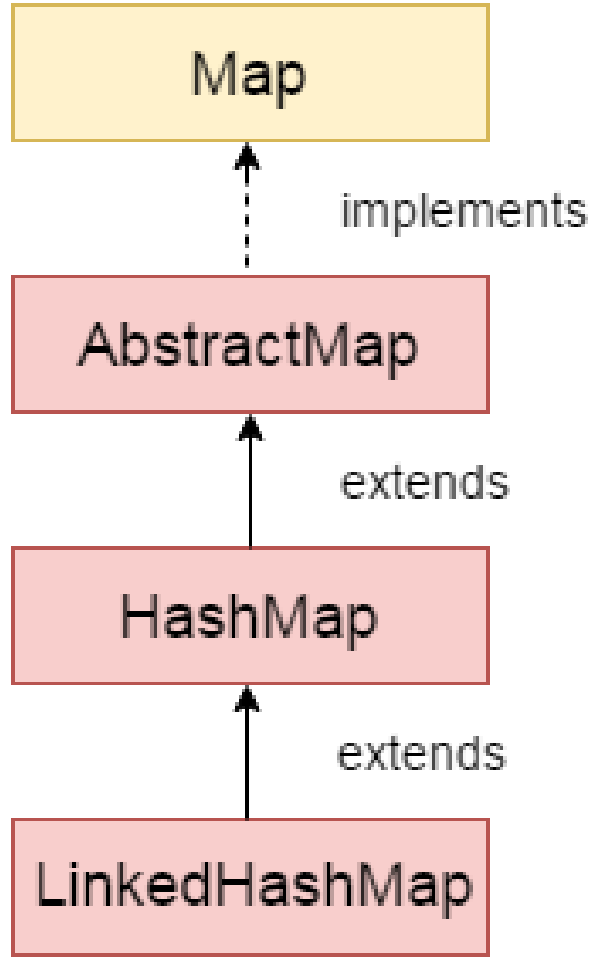
```java
package Mapcollection;
import java.util.Map;
import java.util.HashMap;
public class map1 {

    public static void main(String[] args) {
        // Creating a map using the HashMap
        Map<String, Integer> numbers = new HashMap<>();

        // Insert elements to the map
        numbers.put("One", 1);
        numbers.put("Two", 2);
        System.out.println("Map: " + numbers);

        // Access keys of the map
        System.out.println("Keys: " + numbers.keySet());

        // Access values of the map
        System.out.println("Values: " + numbers.values());

        // Access entries of the map
        System.out.println("Entries: " + numbers.entrySet());

        // Remove Elements from the map
        int value = numbers.remove("Two");
        System.out.println("Removed Value: " + value);

    }

}
```

Output

```
Map: {One=1, Two=2}
Keys: [One, Two]
Values: [1, 2]
Entries: [One=1, Two=2]
Removed Value: 2
```

LinkedHashMap

- A linkedHashmap contains values based on the key. It implements the Map interface and extends HashMap class
- It has only unique elements
- It may have one null key and multiple null values
- It is same as HashMap instead maintains insertion order



TreeMap class Parameters

- Parameters for java.util.TreeMap class.
  - **K**: It is the type of keys maintained by this map.
  - **V**: It is the type of mapped values.
- Some methods are

Map.Entry<K,V> ceilingEntry(K key)	It returns the key-value pair having the least key, greater than or equal to the specified key, or null if there is no such key.
K ceilingKey(K key)	It returns the least key, greater than the specified key or null if there is no such key.
void clear()	It removes all the key-value pairs from a map.
Object clone()	It returns a shallow copy of TreeMap instance.

Method	Description
V get(Object key)	It returns the value to which the specified key is mapped.
void clear()	It removes all the key-value pairs from a map.
boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
Set<Map.Entry<K,V>> entrySet()	It returns a Set view of the mappings contained in the map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped or defaultValue if this map contains no mapping for the key.
Set<K> keySet()	It returns a Set view of the keys contained in the map
protected boolean removeEldestEntry(Map.Entry<K,V> eldest)	It returns true on removing its eldest entry.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It 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.
Collection<V> values()	It returns a Collection view of the values contained in this map.

linkedHashMap sample program

```java
package Mapcollection;
import java.util.*;
public class linkedhashMap1 {

    public static void main(String[] args) {
        LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();
          map.put(100,"Amit");
         map.put(101,"Vijay");
         map.put(102,"Rahul");

         for(Map.Entry m:map.entrySet()){
               System.out.println(m.getKey()+" "+m.getValue());
              }
           //Fetching key
           System.out.println("Keys: "+map.keySet());
           //Fetching value
           System.out.println("Values: "+map.values());
           //Fetching key-value pair
           System.out.println("Key-Value pairs: "+map.entrySet());

           System.out.println("Before invoking remove() method: "+map);
           map.remove(102);
           System.out.println("After invoking remove() method: "+map);
         }
        }
```

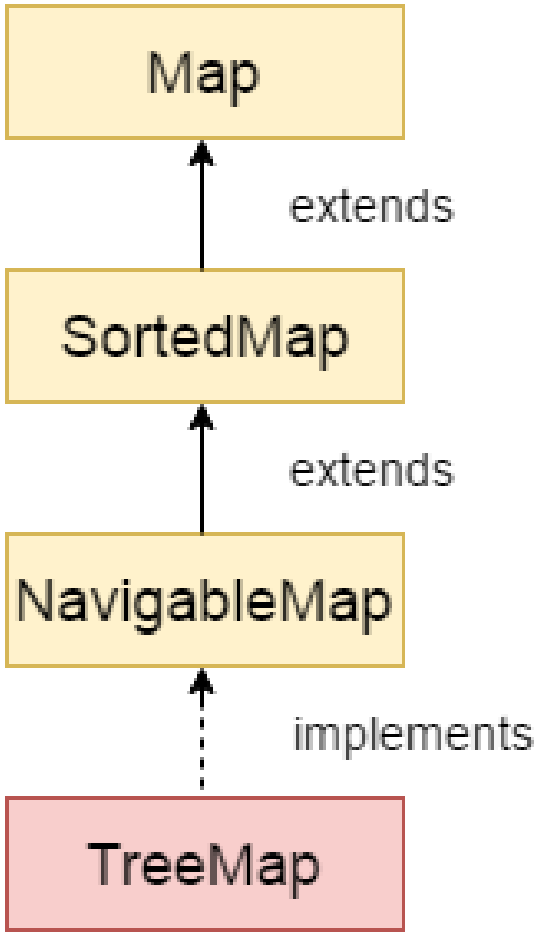
Output

```
100 Amit
101 Vijay
102 Rahul
Keys: [100, 101, 102]
Values: [Amit, Vijay, Rahul]
Key-Value pairs: [100=Amit, 101=Vijay, 102=Rahul]
Before invoking remove() method: {100=Amit,
101=Vijay, 102=Rahul}
After invoking remove() method: {100=Amit, 101=Vijay
```

TreeMap Class

- The TreeMap class implements the Map interface by using a tree. A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.
- Unlike a HashMap, a TreeMap guarantees that its elements will be sorted in an ascending key order.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.

- Difference between HashMap and TreeMap
  - HashMap can contain null value but TreeMap cannot have null key
  - HashMap maintains no order. TreeMap maintains ascending order



```java
package Mapcollection;
import java.util.Map;
import java.util.TreeMap;
public class TreeMap1 {

    public static void main(String[] args) {
        // Creating Map using TreeMap
        Map<String, Integer> values = new TreeMap<>();

        // Insert elements to map
        values.put("Second", 2);
        values.put("First", 1);
        System.out.println("Map using TreeMap: " + values);

        // Replacing the values
        values.replace("First", 11);
        values.replace("Second", 22);
        System.out.println("New Map: " + values);

        // Remove elements from the map
        int removedValue = values.remove("First");
        System.out.println("Removed Value: " + removedValue);
        System.out.println("Map using TreeMap: " + values);

    }

}
```

Output

```
Map using TreeMap: {First=1, Second=2}
New Map: {First=11, Second=22}
Removed Value: 11
Map using TreeMap: {Second=22}
```

Java Hashtable class

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.
- Java Hashtable class is synchronized.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

```java
package Mapcollection;
import java.util.*;
public class hashtable1 {

    public static void main(String[] args) {
        Hashtable<Integer,String> map=new Hashtable<Integer,String>();
         map.put(100,"Amit");
         map.put(102,"Ravi");
         map.put(101,"Vijay");
         map.put(103,"Rahul");
         System.out.println("Before remove: "+ map);
           // Remove value for key 102
           map.remove(102);
           System.out.println("After remove: "+ map);
         //Here, we specify the if and else statement as arguments of the method
           System.out.println(map.getOrDefault(101, "Not Found"));
           System.out.println(map.getOrDefault(105, "Not Found"));
         //Inserts, as the specified pair is unique
           map.putIfAbsent(104,"Gaurav");
           System.out.println("Updated Map: "+map);
           //Returns the current value, as the specified pair already exist
           map.putIfAbsent(101,"Vijay");
           System.out.println("Updated Map: "+map);

    }

}
```

Output

```
Before remove: {103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
After remove: {103=Rahul, 101=Vijay, 100=Amit}
Vijay
Not Found
Updated Map: {104=Gaurav, 103=Rahul, 101=Vijay, 100=Amit}
Updated Map: {104=Gaurav, 103=Rahul, 101=Vijay, 100=Amit}
```

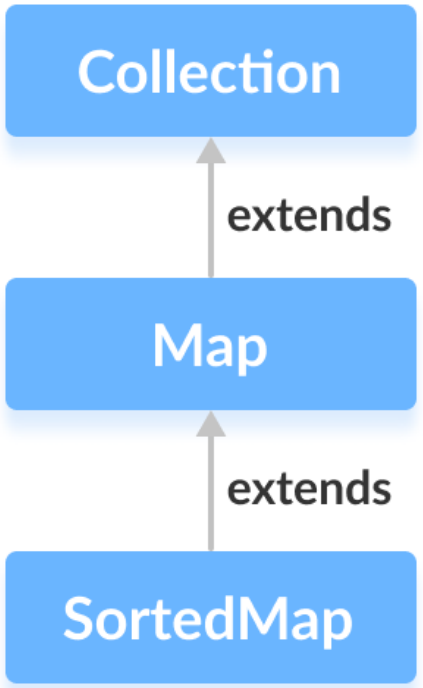
HashMap vs Hashtable

HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

SortedMap

- The SortedMap interface of the Java collections framework provides sorting of keys stored in a map.

- To use the SortedMap, we must import the java.util.SortedMap package first. Once we import the package, here's how we can create a sorted map.




```
// SortedMap implementation by TreeMap class
SortedMap<Key, Value> numbers = new TreeMap<>();
```

Methods of SortedMap

- **comparator()** - returns a comparator that can be used to order keys in a map
- **firstKey()** - returns the first key of the sorted map
- **lastKey()** - returns the last key of the sorted map
- **headMap(key)** - returns all the entries of a map whose keys are less than the specified key
- **tailMap(key)** - returns all the entries of a map whose keys are greater than or equal to the specified key
- **subMap(key1, key2)** - returns all the entries of a map whose keys lies in between key1 and key2 including key1

SortedMap in TreeMap Class

```java
package hashset1;
import java.util.SortedMap;
import java.util.TreeMap;
public class sortedSet {

    public static void main(String[] args) {
        // Creating SortedMap using TreeMap
        SortedMap<String, Integer> numbers = new TreeMap<>();

        // Insert elements to map
        numbers.put("Two", 2);
        numbers.put("One", 1);
        System.out.println("SortedMap: " + numbers);


        // Access the first key of the map
        System.out.println("First Key: " + numbers.firstKey());

        // Access the last key of the map
        System.out.println("Last Key: " + numbers.lastKey());

        // Remove elements from the map
        int value = numbers.remove("One");
        System.out.println("Removed Value: " + value);

    }

}
```

Output

```
SortedMap: {One=1, Two=2}
First Key: One
Last Key: Two
Removed Value: 1
```

Namah Shivaya