Java Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:

Java Map Hierarchy

A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

A Map can't be traversed, so you need to convert it into Set using *keySet()* or *entrySet()* method.

|  |  |
| --- | --- |
| **Class** | **Description** |
| [HashMap](https://www.javatpoint.com/java-hashmap) | HashMap is the implementation of Map, but it doesn't maintain  any order. |
| [LinkedHashMap](https://www.javatpoint.com/java-linkedhashmap) | LinkedHashMap is the implementation of Map. It inherits HashMap  class. It maintains insertion order. |
| [TreeMap](https://www.javatpoint.com/java-treemap) | TreeMap is the implementation of Map and SortedMap.  It maintains ascending order. |

Useful methods of Map interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| V put(Object key, Object value) | It is used to insert  an entry in the map. |
| void putAll(Map map) | It is used to insert  the specified  map in the map. |
| V putIfAbsent(K key, V value) | It inserts the specified  value with the  specified key in  the map only  if it is not  already specified. |
| V remove(Object key) | It is used to delete an  entry for the specified key. |
| boolean remove(Object key, Object value) | It removes the specified  values with the associated  specified keys  from the map. |
| Set keySet() | It returns the Set view  containing all the keys. |
| Set<Map.Entry<K,V>> entrySet() | It returns the Set view  containing all the keys  and values. |
| void clear() | It is used to reset the map. |
| V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) | It is used to compute  a mapping  for the specified key  and its  current mapped value  (or null if there is  no current mapping). |
| V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction) | It is used to compute its  value using the given  mapping function,  if the specified  key is not already  associated  with a value  (or is mapped to  null), and enters it  into this map  unless null. |
| V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) | It is used to compute a  new mapping  given the key  and its current mapped  value if the value  for the specified key  is present and non-null. |
| boolean containsValue(Object value) | This method returns true if some  value equal to the value exists  within the map,  else return false. |
| boolean containsKey(Object key) | This method  returns true if some  key equal to the key exists within  the map, else return false. |
| boolean equals(Object o) | It is used to compare  the specified  Object with 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 get(Object key) | This method returns the  object  that contains the value  associated  with the key. |
| V getOrDefault(Object key, V defaultValue) | It returns the value to  which  the specified key is  mapped,  or defaultValue if the  map  contains no  mapping for the key. |
| int hashCode() | It returns the hash code  value for the Map |
| boolean isEmpty() | This method returns true if the map is empty; returns false if it contains at least one key. |
| V 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, associates  it with the given  non-null value. |
| V replace(K key, V value) | It replaces the s  pecified value  for a specified key. |
| boolean replace(K key, V oldValue, V newValue) | It replaces the old  value with  the new value for a  specified key. |
| 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 values() | It returns a collection  view of the values  contained in the map. |
| int size() | This method returns  the number of entries  in the map. |

Map.Entry Interface

Entry is the subinterface of Map. So we will be accessed it by Map.Entry name. It returns a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

Methods of Map.Entry interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| K getKey() | It is used to  obtain a  key. |
| V getValue() | It is used to  obtain  value. |
| int hashCode() | It is used to  obtain  hashCode. |
| V setValue(V value) | It is used to  replace the  value  corresponding to this entry with the specified value. |
| boolean equals(Object o) | It is used to  compare the specified object with the other existing objects. |
| static <K extends Comparable<? super K>,V> Comparator<Map.Entry<K,V>> comparingByKey() | It returns a  comparator that  compare the  objects  in natural order  on  key. |
| static <K,V> Comparator<Map.Entry<K,V>> comparingByKey(Comparator<? super K> cmp) | It returns a  comparator that  compare the  objects by key  using the given  Comparator. |
| static <K,V extends Comparable<? super V>> Comparator<Map.Entry<K,V>> comparingByValue() | It returns a  comparator that  compare the  objects  in natural order  on  value. |
| static <K,V> Comparator<Map.Entry<K,V>> comparingByValue(Comparator<? super V> cmp) | It returns a  Comparator  that compare  the objects by  value using the given Comparator. |

Java Map Example: Non-Generic (Old Style)

1. //Non-generic
2. **import** java.util.\*;
3. **public** **class** MapExample1 {
4. **public** **static** **void** main(String[] args) {
5. Map map=**new** HashMap();
6. //Adding elements to map
7. map.put(1,"Amit");
8. map.put(5,"Rahul");
9. map.put(2,"Jai");
10. map.put(6,"Amit");
11. //Traversing Map
12. Set set=map.entrySet();//Converting to Set so that we can traverse
13. Iterator itr=set.iterator();
14. **while**(itr.hasNext()){
15. //Converting to Map.Entry so that we can get key and value separately
16. Map.Entry entry=(Map.Entry)itr.next();
17. System.out.println(entry.getKey()+" "+entry.getValue());
18. }
19. }
20. }

Output:

1 Amit

2 Jai

5 Rahul

6 Amit

Java Map Example: Generic (New Style)

1. **import** java.util.\*;
2. **class** MapExample2{
3. **public** **static** **void** main(String args[]){
4. Map<Integer,String> map=**new** HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. //Elements can traverse in any order
9. **for**(Map.Entry m:map.entrySet()){
10. System.out.println(m.getKey()+" "+m.getValue());
11. }
12. }
13. }

Output:

102 Rahul

100 Amit

101 Vijay

Java Map Example: comparingByKey()

1. **import** java.util.\*;
2. **class** MapExample3{
3. **public** **static** **void** main(String args[]){
4. Map<Integer,String> map=**new** HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. //Returns a Set view of the mappings contained in this map
9. map.entrySet()
10. //Returns a sequential Stream with this collection as its source
11. .stream()
12. //Sorted according to the provided Comparator
13. .sorted(Map.Entry.comparingByKey())
14. //Performs an action for each element of this stream
15. .forEach(System.out::println);
16. }
17. }

Output:

100=Amit

101=Vijay

102=Rahul

Java Map Example: comparingByKey() in Descending Order

1. **import** java.util.\*;
2. **class** MapExample4{
3. **public** **static** **void** main(String args[]){
4. Map<Integer,String> map=**new** HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. //Returns a Set view of the mappings contained in this map
9. map.entrySet()
10. //Returns a sequential Stream with this collection as its source
11. .stream()
12. //Sorted according to the provided Comparator
13. .sorted(Map.Entry.comparingByKey(Comparator.reverseOrder()))
14. //Performs an action for each element of this stream
15. .forEach(System.out::println);
16. }
17. }

Output:

102=Rahul

101=Vijay

100=Amit

Java Map Example: comparingByValue()

1. **import** java.util.\*;
2. **class** MapExample5{
3. **public** **static** **void** main(String args[]){
4. Map<Integer,String> map=**new** HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. //Returns a Set view of the mappings contained in this map
9. map.entrySet()
10. //Returns a sequential Stream with this collection as its source
11. .stream()
12. //Sorted according to the provided Comparator
13. .sorted(Map.Entry.comparingByValue())
14. //Performs an action for each element of this stream
15. .forEach(System.out::println);
16. }
17. }

Output:

100=Amit

102=Rahul

101=Vijay

Java Map Example: comparingByValue() in Descending Order

1. **import** java.util.\*;
2. **class** MapExample6{
3. **public** **static** **void** main(String args[]){
4. Map<Integer,String> map=**new** HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. //Returns a Set view of the mappings contained in this map
9. map.entrySet()
10. //Returns a sequential Stream with this collection as its source
11. .stream()
12. //Sorted according to the provided Comparator
13. .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
14. //Performs an action for each element of this stream
15. .forEach(System.out::println);
16. }
17. }

Output:

101=Vijay

102=Rahul

100=Amit