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 specified 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)

//Non-generic

**import** java.util.\*;

**public** **class** MapExample1 {

**public** **static** **void** main(String[] args) {

    Map map=**new** HashMap();

    //Adding elements to map

    map.put(1,"Amit");

   map.put(5,"Rahul");

    map.put(2,"Jai");

    map.put(6,"Amit");

   //Traversing Map

    Set set=map.entrySet();//Converting to Set so that we can traverse

    Iterator itr=set.iterator();

**while**(itr.hasNext()){

        //Converting to Map.Entry so that we can get key and value separately

        Map.Entry entry=(Map.Entry)itr.next();

        System.out.println(entry.getKey()+" "+entry.getValue());

    }

}

}

### Java Map Example: Generic (New Style)

**import** java.util.\*;

**class** MapExample2{

**public** **static** **void** main(String args[]){

  Map<Integer,String> map=**new** HashMap<Integer,String>();

  map.put(100,"Amit");

  map.put(101,"Vijay");

  map.put(102,"Rahul");

  //Elements can traverse in any order

**for**(Map.Entry m:map.entrySet()){

   System.out.println(m.getKey()+" "+m.getValue());

  }

 }

}

# Java HashMap



Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as HashMap<K,V>, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

### Points to remember

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.
* The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

### Hierarchy of HashMap class

  Check if a particular key exists in Java HashMap example

  This Java Example shows how to check if HashMap object contains a particular

  key using containsKey method of HashMap class.

\*/

import java.util.HashMap;

public class CheckKeyOfHashMapExample {

  public static void main(String[] args) {

    //create HashMap object

    HashMap hMap = new HashMap();

    //add key value pairs to HashMap

    hMap.put("1","One");

    hMap.put("2","Two");

    hMap.put("3","Three");

    /\*

      To check whether a particular key exists in HashMap use

      boolean containsKey(Object key) method of HashMap class.

      It returns true if the HashMap contains mapping for specified key

      otherwise false.

    \*/

    boolean blnExists = hMap.containsKey("3");

    System.out.println("3 exists in HashMap ? : " + blnExists);

  }

}

/\*

  Get Set view of Keys from Java HashMap example

  This Java Example shows how to get a Set of keys contained in HashMap

  using keySet method of Java HashMap class.

\*/

import java.util.Iterator;

import java.util.HashMap;

import java.util.Set;

public class GetSetViewOfKeysFromHashMapExample {

  public static void main(String[] args) {

    //create HashMap object

    HashMap hMap = new HashMap();

    //add key value pairs to HashMap

    hMap.put("1","One");

    hMap.put("2","Two");

    hMap.put("3","Three");

    /\*

      get Set of keys contained in HashMap using

      Set keySet() method of HashMap class

    \*/

    Set st = hMap.keySet();

    System.out.println("Set created from HashMap Keys contains :");

    //iterate through the Set of keys

    Iterator itr = st.iterator();

    while(itr.hasNext())

      System.out.println(itr.next());

    /\*

       Please note that resultant Set object is backed by the HashMap.

       Any key that is removed from Set will also be removed from

       original HashMap object. The same is not the case with the element

       addition.

    \*/

    //remove 2 from Set

    st.remove("2");

    //check if original HashMap still contains 2

    boolean blnExists = hMap.containsKey("2");

    System.out.println("Does HashMap contain 2 ? " + blnExists);

  }

}

/\*

  Check if a particular key exists in Java LinkedHashMap example

  This Java Example shows how to check if LinkedHashMap object contains a particular

  key using containsKey method of LinkedHashMap class.

\*/

import java.util.LinkedHashMap;

public class CheckKeyOfLinkedHashMapExample {

  public static void main(String[] args) {

    //create LinkedHashMap object

    LinkedHashMap lHashMap = new LinkedHashMap();

    //add key value pairs to LinkedHashMap

    lHashMap.put("1","One");

    lHashMap.put("2","Two");

    lHashMap.put("3","Three");

    /\*

      To check whether a particular key exists in LinkedHashMap use

      boolean containsKey(Object key) method of LinkedHashMap class.

      It returns true if the LinkedHashMap contains mapping for specified key

      otherwise false.

    \*/

    boolean blnExists = lHashMap.containsKey("3");

    System.out.println("3 exists in LinkedHashMap ? : " + blnExists);

  }

}

/\*

  Get Size of Java LinkedHashMap Example

  This Java Example shows how to get the size or nubmer of key value pairs

  stored in LinkedHashMap using size method.

\*/

import java.util.LinkedHashMap;

public class GetSizeOfLinkedHashMapExample{

  public static void main(String[] args) {

    //create LinkedHashMap object

    LinkedHashMap lHashMap = new LinkedHashMap();

    /\*

      To get the size of LinkedHashMap use

      int size() method of LinkedHashMap class. It returns the number of key value

      pairs stored in LinkedHashMap object.

    \*/

    System.out.println("Size of LinkedHashMap : " + lHashMap.size());

    //add key value pairs to LinkedHashMap using put method

    lHashMap.put("1","One");

    lHashMap.put("2","Two");

    lHashMap.put("3","Three");

    System.out.println("Size of LinkedHashMap after addition : " + lHashMap.size());

    //remove one element from LinkedHashMap using remove method

    Object obj = lHashMap.remove("2");

    System.out.println("Size of LinkedHashMap after removal : " + lHashMap.size());

  }

}

/\*

  Check if a particular key exists in Java TreeMap example

  This Java Example shows how to check if TreeMap object contains a particular

  key using containsKey method of TreeMap class.

\*/

import java.util.TreeMap;

public class CheckKeyOfTreeMapExample {

  public static void main(String[] args) {

    //create TreeMap object

    TreeMap treeMap = new TreeMap();

    //add key value pairs to TreeMap

    treeMap.put("1","One");

    treeMap.put("2","Two");

    treeMap.put("3","Three");

    /\*

      To check whether a particular key exists in TreeMap use

      boolean containsKey(Object key) method of TreeMap class.

      It returns true if the TreeMap contains mapping for specified key

      otherwise false.

    \*/

    boolean blnExists = treeMap.containsKey("1");

    System.out.println("1 exists in TreeMap ? : " + blnExists);

  }

}

/\*

  Get Size of Java TreeMap Example

  This Java Example shows how to get the size or nubmer of key value pairs

  stored in TreeMap using size method.

\*/

import java.util.TreeMap;

public class GetSizeOfTreeMapExample{

  public static void main(String[] args) {

    //create TreeMap object

    TreeMap treeMap = new TreeMap();

    /\*

      To get the size of TreeMap use

      int size() method of TreeMap class. It returns the number of key value

      pairs stored in TreeMap object.

    \*/

    System.out.println("Size of TreeMap : " + treeMap.size());

    //add key value pairs to TreeMap using put method

    treeMap.put("1","One");

    treeMap.put("2","Two");

    treeMap.put("3","Three");

    System.out.println("Size of TreeMap after addition : " + treeMap.size());

    //remove one element from TreeMap using remove method

    Object obj = treeMap.remove("2");

    System.out.println("Size of TreeMap after removal : " + treeMap.size());

  }

}

# How to Sort HashMap in Java

Java HashMap does not preserve any order by default. If there is a need to sort HashMap we sort it explicitly based on the requirements. Java provides an option to sort HashMap based on keys and values. In this section, we will learn how to sort HashMap according to keys and values.

* Sort HashMap by Keys
* Sort HashMap by Values

## Sort HashMap by Keys

There are following ways to sort HashMap by keys:

* By using **TreeMap**
* By using **LinkedHashMap**

When we use LinkedHashMap, we should follow the process:

**comparator**

/\*

  Sort Java ArrayList in descending order using comparator example

  This java example shows how to sort elements of Java ArrayList in descending order

  using comparator and reverseOrder method of Collections class.

\*/

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

public class SortArrayListInDescendingOrderExample {

  public static void main(String[] args) {

    //create an ArrayList object

    ArrayList arrayList = new ArrayList();

    //Add elements to Arraylist

    arrayList.add("A");

    arrayList.add("B");

    arrayList.add("C");

    arrayList.add("D");

    arrayList.add("E");

    /\*

      To get comparator that imposes reverse order on a Collection use

      static Comparator reverseOrder() method of Collections class

    \*/

    Comparator comparator = Collections.reverseOrder();

    System.out.println("Before sorting ArrayList in descending order : "

                                                              + arrayList);

    /\*

      To sort an ArrayList using comparator use,

      static void sort(List list, Comparator c) method of Collections class.

    \*/

    Collections.sort(arrayList,comparator);

    System.out.println("After sorting ArrayList in descending order : "

                                                              + arrayList);

  }

}

/\*

Output would be

Before sorting ArrayList in descending order : [A, B, C, D, E]

After sorting ArrayList in descending order : [E, D, C, B, A]

\*/

/\*

  Sort Java Vector in descending order using comparator example

  This java example shows how to sort elements of Java Vector in descending order

  using comparator and reverseOrder method of Collections class.

\*/

import java.util.Vector;

import java.util.Collections;

import java.util.Comparator;

public class SortVectorInDescendingOrderExample {

  public static void main(String[] args) {

    //create a Vector object

    Vector v = new Vector();

    //Add elements to Vector

    v.add("1");

    v.add("2");

    v.add("3");

    v.add("4");

    v.add("5");

    /\*

      To get comparator that imposes reverse order on a Collection use

      static Comparator reverseOrder() method of Collections class

    \*/

    Comparator comparator = Collections.reverseOrder();

    System.out.println("Before sorting Vector in descending order : " + v);

    /\*

      To sort an Vector using comparator use,

      static void sort(List list, Comparator c) method of Collections class.

    \*/

    Collections.sort(v,comparator);

    System.out.println("After sorting Vector in descending order : " + v);

  }

}

/\*

Output would be

Before sorting Vector in descending order : [1, 2, 3, 4, 5]

After sorting Vector in descending order : [5, 4, 3, 2, 1]

\*/