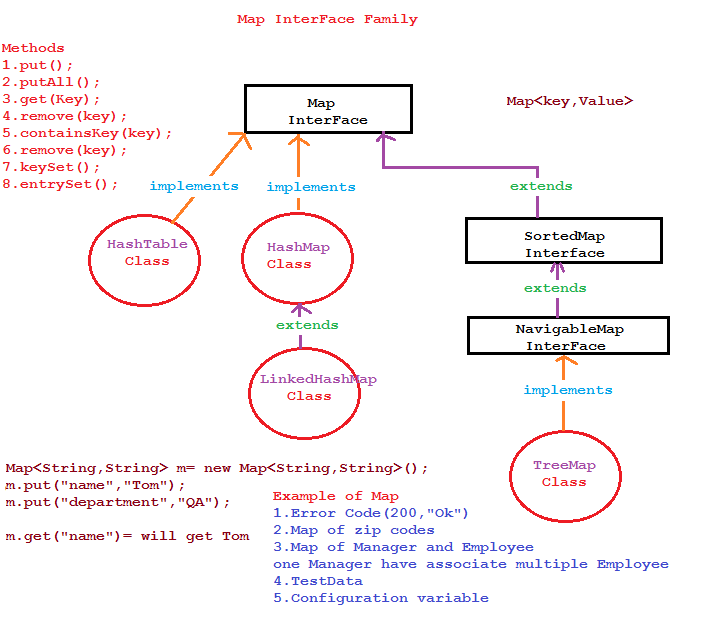
# Java Map Interface

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.

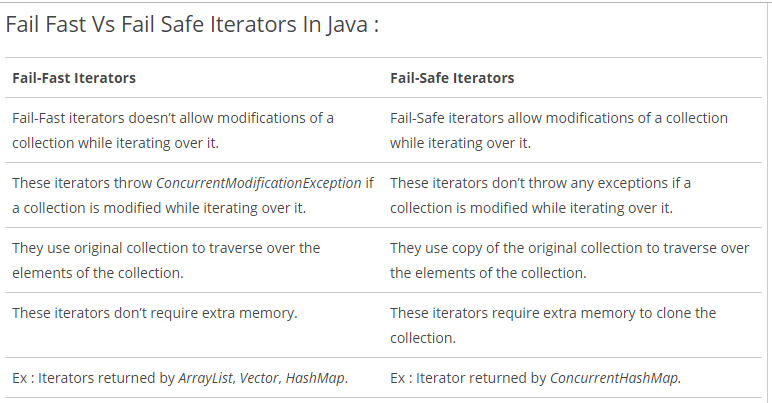
|  |  |
| --- | --- |
| **Class** | **Description** |
| [HashMap](https://www.javatpoint.com/java-hashmap) | HashMap is the implementation of Map, but it doesn't maintain any order. |
| [LinkedHashMap](https://www.javatpoint.com/java-linkedhashmap) | LinkedHashMap is the implementation of Map. It inherits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

## Map.Entry Interface

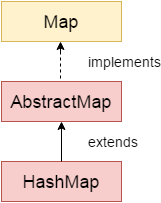
Entry is the subinterface of Map. So we will be accessed it by Map.Entry name. It returns a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

Dictionary class?

The Dictionary class provides the capability to store key-value pairs.



# Java HashMap class



Java HashMap class implements the map interface by using a hash table. It inherits AbstractMap class and implements Map interface.

### Points to remember

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

### Hierarchy of HashMap class

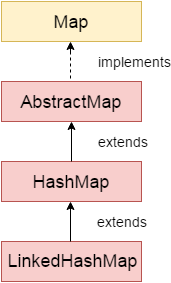
As shown in the above figure, HashMap class extends AbstractMap class and implements Map interface.

### HashMap class declaration

Let's see the declaration for java.util.HashMap class.

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

# Java LinkedHashMap class



Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

### Points to remember

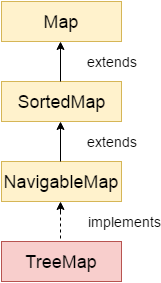
* Java LinkedHashMap contains values based on the key.
* Java LinkedHashMap contains unique elements.
* Java LinkedHashMap may have one null key and multiple null values.
* Java LinkedHashMap is non synchronized.
* Java LinkedHashMap maintains insertion order.
* The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

LinkedHashMap class Parameters

Let's see the Parameters for java.util.LinkedHashMap class.

* **K**: It is the type of keys maintained by this map.
* **V**: It is the type of mapped values.

# Java TreeMap class



Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

* Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* Java TreeMap contains only unique elements.
* Java TreeMap cannot have a null key but can have multiple null values.
* Java TreeMap is non synchronized.
* Java TreeMap maintains ascending order.

### TreeMap class declaration

Let's see the declaration for java.util.TreeMap class.

1. **public** **class** TreeMap<K,V> **extends** AbstractMap<K,V> **implements** NavigableMap<K,V>, Cloneable, Serializable

### TreeMap class Parameters

Let's see the 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.
* What is difference between HashMap and TreeMap?

|  |  |
| --- | --- |
| **HashMap** | **TreeMap** |
| 1) HashMap can contain one null key. | TreeMap cannot contain any null key. |
| 2) HashMap maintains no order. | TreeMap maintains ascending order. |

# Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

### Points to remember

* 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.

### Hashtable class declaration

Let's see the declaration for java.util.Hashtable class.

1. **public** **class** Hashtable<K,V> **extends** Dictionary<K,V> **implements** Map<K,V>, Cloneable, Serializable

### Hashtable class Parameters

Let's see the Parameters for java.util.Hashtable class.

* **K**: It is the type of keys maintained by this map.
* **V**: It is the type of mapped values.

# Difference between HashMap and 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. |

# Sorting in Collection

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

|  |
| --- |
| **Collections** class provides static methods for sorting the elements of a collection. If collection elements are of a Set type, we can use TreeSet. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements. |

### Method of Collections class for sorting List elements

**public void sort(List list):** is used to sort the elements of List. List elements must be of the Comparable type.

#### Note: String class and Wrapper classes implement the Comparable interface. So if you store the objects of string or wrapper classes, it will be Comparable.

# Java Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.

### compareTo(Object obj) method

**public int compareTo(Object obj):** It is used to compare the current object with the specified object. It returns

* positive integer, if the current object is greater than the specified object.
* negative integer, if the current object is less than the specified object.
* zero, if the current object is equal to the specified object.
* **class** Student **implements** Comparable<Student>{
* **int** rollno;
* String name;
* **int** age;
* Student(**int** rollno,String name,**int** age){
* **this**.rollno=rollno;
* **this**.name=name;
* **this**.age=age;
* }
* **public** **int** compareTo(Student st){
* **if**(age==st.age)
* **return** 0;
* **else** **if**(age>st.age)
* **return** 1;
* **else**
* **return** -1;
* }
* }

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

### Collections class

**Collections** class provides static methods for sorting the elements of collections. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

### Method of Collections class for sorting List elements

**public void sort(List list):** It is used to sort the elements of List. List elements must be of the Comparable type.

#### Note: String class and Wrapper classes implement the Comparable interface by default. So if you store the objects of string or wrapper classes in a list, set or map, it will be Comparable by default.

# Java Comparator interface

**Java Comparator interface** is used to order the objects of a user-defined class.

This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

## Collections class

**Collections** class provides static methods for sorting the elements of a collection. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements also.

# Properties class in Java

The **properties** object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from the properties file and store data into the properties file. Moreover, it can be used to get the properties of a system.

An Advantage of the properties file

**Recompilation is not required if the information is changed from a properties file:** If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.