Map Interface in Java

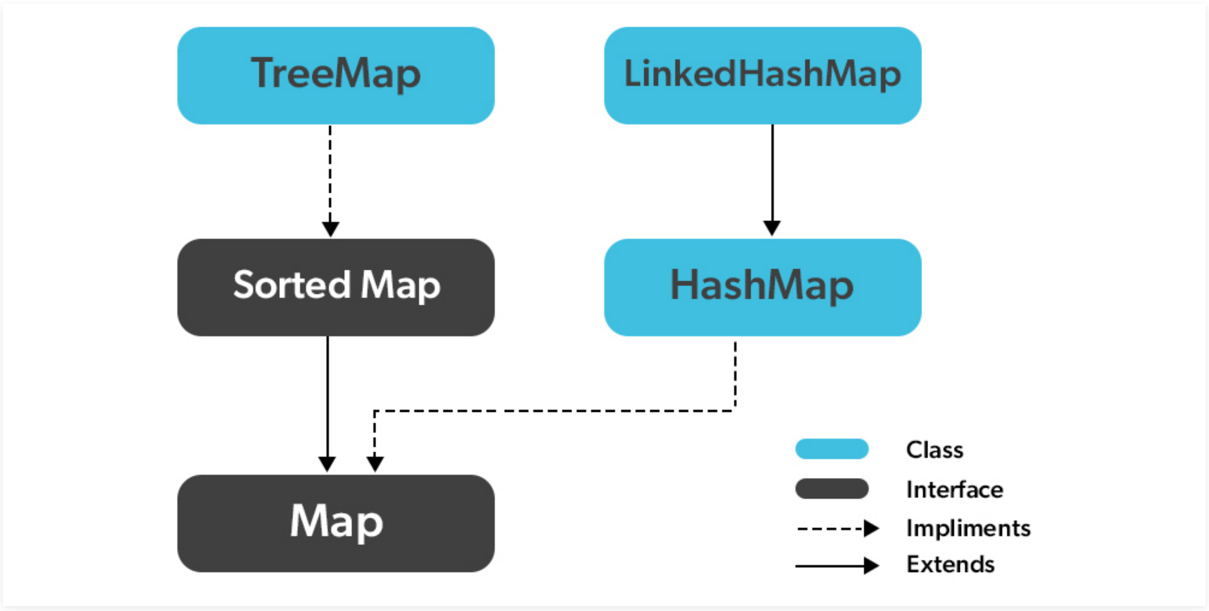
Difficulty Level : Easy Last Updated : 24 Jan, 2022

The map interface is present in java.util package represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.

Geeks, the brainstormer should have been **why and when to use Maps?**

Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys. Some common scenarios are as follows:

- A map of error codes and their descriptions.
- A map of zip codes and cities.
- A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
- A map of classes and students. Each class (key) is associated with a list of students (value).



Creating Map Objects

Since Map is an interface, objects cannot be created of the type map. We always need a class that extends this map in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the Map.

Syntax: Defining Type-safe Map

```
Map hm = new HashMap();
// Obj is the type of the object to be stored in Map
```

Characteristics of a Map Interface

1. A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null values like the HashMap and LinkedHashMap, but some do not like the TreeMap.
2. The order of a map depends on the specific implementations. For example, TreeMap and LinkedHashMap have predictable orders, while HashMap does not.
3. There are two interfaces for implementing Map in java. They are Map and SortedMap, and three classes: HashMap, TreeMap, and LinkedHashMap.

Methods in Map Interface

Method	Action Performed
clear()	This method is used to clear and remove all of the elements or mappings from a specified Map collection.
containsKey(Object)	This method is used 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 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 key in the map.
entrySet()	This method is used 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 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.

Method	Action Performed
hashCode()	This method is used 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.
keySet()	This method is used 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 to associate the specified value with the specified key in this map.
putAll(Map)	This method is used to copy all of the mappings from the specified map to this map.
remove(Object)	This method is used 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 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, associates 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 curassociaterent value.

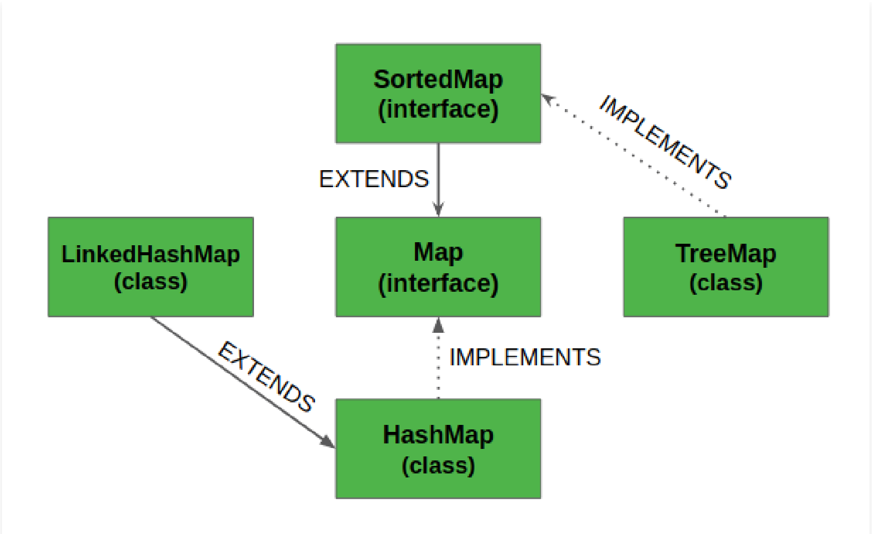
Example:

```java
// Java Program to Demonstrate
// Working of Map interface

// Importing required classes
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an empty HashMap
        Map<String, Integer> hm
            = new HashMap<String, Integer>();

        // Inserting pairs in above Map
        // using put() method
        hm.put("a", new Integer(100));
        hm.put("b", new Integer(200));
        hm.put("c", new Integer(300));
        hm.put("d", new Integer(400));

        // Traversing through Map using for-each loop
        for (Map.Entry<String, Integer> me :
             hm.entrySet()) {

            // Printing keys
            System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

Output:

```
a:100
b:200
c:300
d:400
```

Classes that implement the Map interface are depicted in the below media and described later as follows:



MAP Hierarchy in Java

Class 1: HashMap

HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key. This class uses a technique called Hashing. Hashing is a technique of converting a large String to a small String that represents the same String. A shorter value helps in indexing and faster searches. Let's see how to create a map object using this class.

Example

```
// Java Program to illustrate the Hashmap Class

// Importing required classes
import java.util.*;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating an empty HashMap
        Map<String, Integer> map = new HashMap<>();

        // Inserting entries in the Map
        // using put() method
        map.put("vishal", 10);
        map.put("sachin", 30);
        map.put("vaibhav", 20);
        // Iterating over Map
        for (Map.Entry<String, Integer> e : map.entrySet())

            // Printing key-value pairs
            System.out.println(e.getKey() + " "
                               + e.getValue());
    }
}
```

Output

```
vaibhav 20
vishal 10
sachin 30
```

Class 2: LinkedHashMap
LinkedHashMap is just like HashMap with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion which the LinkedHashMap provides where the elements can be accessed in their insertion order. Let’s see how to create a map object using this class.

Example

```
// Java Program to Illustrate the LinkedHashmap Class

// Importing required classes
import java.util.*;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating an empty LinkedHashMap
        Map<String, Integer> map = new LinkedHashMap<>();

        // Inserting pair entries in above Map
        // using put() method
        map.put("vishal", 10);
        map.put("sachin", 30);
        map.put("vaibhav", 20);

        // Iterating over Map
        for (Map.Entry<String, Integer> e : map.entrySet())

            // Printing ket-value pairs
            System.out.println(e.getKey() + " "
                               + e.getValue());
    }
}
```

Output:

```
vishal 10
sachin 30
vaibhav 20
```

Class 3: TreeMap
The TreeMap in Java is used to implement the Map interface and NavigableMap along with the Abstract Class. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used. This proves to be an efficient way of sorting and storing the key-value pairs. The storing order maintained by the treemap must be consistent with equals just like any other sorted map, irrespective of the explicit comparators. Let’s see how to create a map object using this class.

Example

```
// Java Program to Illustrate TreeMap Class

// Importing required classes
import java.util.*;

// Main class
public class GFG {
    // Main driver method
    public static void main(String[] args)
    {

        // Creating an empty TreeMap
        Map<String, Integer> map = new TreeMap<>();

        // Inserting custom elements in the Map
        // using put() method
        map.put("vishal", 10);
        map.put("sachin", 30);
        map.put("vaibhav", 20);

        // Iterating over Map using for each loop
        for (Map.Entry<String, Integer> e : map.entrySet())

            // Printing key-value pairs
            System.out.println(e.getKey() + " "
                               + e.getValue());
    }
}
```

Output:

```
sachin 30
vaibhav 20
vishal 10
```

Performing Various Operations using *Map Interface* and *HashMap Class*

Since Map is an interface, it can be used only with a class that implements this interface. Now, let's see how to perform a few frequently used operations on a Map using the widely used HashMap class. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the map.

Operation 1: Adding Elements

In order to add an element to the map, we can use the put() method. However, the insertion order is not retained in the hashmap. Internally, for every element, a separate hash is generated and the elements are indexed based on this hash to make it more efficient.

Example

```java
// Java program to demonstrate
// the working of Map interface

import java.util.*;
class GFG {
    public static void main(String args[])
    {
        // Default Initialization of a
        // Map
        Map<Integer, String> hm1 = new HashMap<>();

        // Initialization of a Map
        // using Generics
        Map<Integer, String> hm2
            = new HashMap<Integer, String>();

        // Inserting the Elements
        hm1.put(1, "Geeks");
        hm1.put(2, "For");
        hm1.put(3, "Geeks");

        hm2.put(new Integer(1), "Geeks");
        hm2.put(new Integer(2), "For");
        hm2.put(new Integer(3), "Geeks");

        System.out.println(hm1);
        System.out.println(hm2);
    }
}
```

Output:

```
{1=Geeks, 2=For, 3=Geeks}
{1=Geeks, 2=For, 3=Geeks}
```

Operation 2: Changing Element

After adding the elements if we wish to change the element, it can be done by again adding the element with the put() method. Since the elements in the map are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.

Example

```java
// Java program to demonstrate
// the working of Map interface

import java.util.*;
class GFG {
    public static void main(String args[])
    {

        // Initialization of a Map
        // using Generics
        Map<Integer, String> hm1
            = new HashMap<Integer, String>();

        // Inserting the Elements
        hm1.put(new Integer(1), "Geeks");
        hm1.put(new Integer(2), "Geeks");
        hm1.put(new Integer(3), "Geeks");

        System.out.println("Initial Map " + hm1);

        hm1.put(new Integer(2), "For");

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

Output:

```
Initial Map {1=Geeks, 2=Geeks, 3=Geeks}
Updated Map {1=Geeks, 2=For, 3=Geeks}
```

Operation 3: Removing Elements

In order to remove an element from the Map, we can use the remove() method. This method takes the key value and removes the mapping for a key from this map if it is present in the map.

Example

```java
// Java program to demonstrate
// the working of Map interface

import java.util.*;
class GFG {

    public static void main(String args[])
    {

        // Initialization of a Map
        // using Generics
        Map<Integer, String> hm1
            = new HashMap<Integer, String>();

        // Inserting the Elements
        hm1.put(new Integer(1), "Geeks");
        hm1.put(new Integer(2), "For");
        hm1.put(new Integer(3), "Geeks");
        hm1.put(new Integer(4), "For");

        // Initial Map
        System.out.println(hm1);

        hm1.remove(new Integer(4));

        // Final Map
        System.out.println(hm1);
    }
}
```

Output:

```
{1=Geeks, 2=For, 3=Geeks, 4=For}
{1=Geeks, 2=For, 3=Geeks}
```

Operation 4: Iterating through the Map

There are multiple ways to iterate through the Map. The most famous way is to use a for-each loop and get the keys. The value of the key is found by using the getValue() method.

Example

```
// Java program to demonstrate
// the working of Map interface

import java.util.*;
class GFG {
    public static void main(String args[])
    {

        // Initialization of a Map
        // using Generics
        Map<Integer, String> hm1
            = new HashMap<Integer, String>();

        // Inserting the Elements
        hm1.put(new Integer(1), "Geeks");
        hm1.put(new Integer(2), "For");
        hm1.put(new Integer(3), "Geeks");

        for (Map.Entry mapElement : hm1.entrySet()) {
            int key
                = (int)mapElement.getKey();

            // Finding the value
            String value
                = (String)mapElement.getValue();

            System.out.println(key + " : "
                               + value);
        }
    }
}
```

Output:

```
1 : Geeks
2 : For
3 : Geeks
```