Maps

- Map is a data structure, which helps the programmer to store the data in the form of key value pairs.
- Where every key is associated with a unique value.

Note:

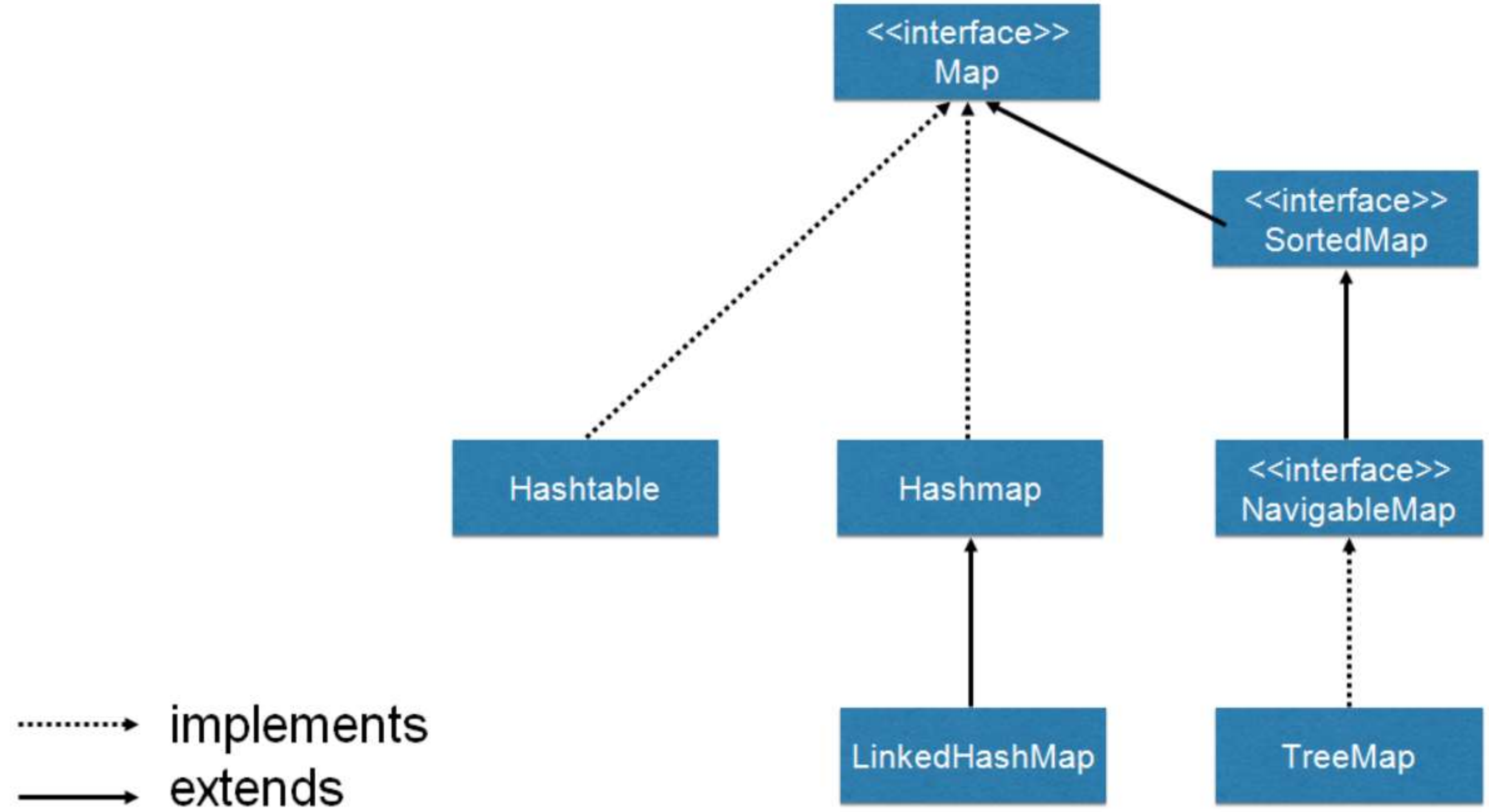
- Key cannot be duplicate
- One key can be associated with only one value.
- Maps helps us to access the values easily with the help of its associated key.
- Map is an interface in java defined in java.util package
- We can create generic map by providing the type for both key as well as value<key-type, value-type>
- We can obtain three different views of a map

a)We can obtain list of values from a map.

b)We can obtain set of keys from a map

c)We can obtain a set of key-value pairs.

Map Interface



How to use Map?

- In Java, we must import the java.util.Map package in order to use Map. Once the package is imported , we can create a map.

- Map implementation using HashMap

```
        Map<Key, Value> map = new HashMap();
```

Here,

In the above code, Map is created named map HashMap class is used to implement the Map interface.

Note: Since Map is an interface, we cannot create object for it.

•Key - a unique identifier used to associate each element (value) in a map

•Value - elements associated by keys in a map

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

Methods in Map Interface

put(key,value)	1)Add entry to the map(key value pair) 2)Replace the old value with a new value of an existing entry in map.
putAll(map)	It will copy all the entries from the given map into the current map
containskey(key)	If the key is present returns true else returns false
containsValue(Value)	If the value is present it returns true, else it returns false
remove(key)	If the key is present the entry is removed from map and the value is returned if the key is not present nothing is removed and null is returned
clear()	It removes all the entries in the map
get(key)	It is used to access the value associated with a particular key. If the key is present it returns value else it returns null
values()	It returns a collection of values present in map

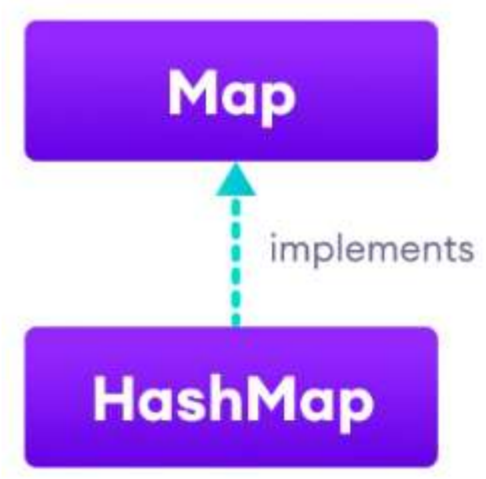
Characteristics of HashMap

HashMap is the concrete implementing class of map interface

The HashMap class of the Java collections framework provides the functionality of the hash table data structure.

Following are the characteristics of HashMap:

- Data is stored in the form of key-value pair
- Order of insertion is not maintained
- Key cannot be duplicate, values can be duplicate.
- Key can be null
- Value can be null



Create a HashMap

In order to create a hash map, we must import the java.util.HashMap package first. Once the package is imported , we can create HashMap in Java.

```
HashMap<K, V> map= new HashMap();
```

- In the above code, HashMap is created named map. Here, **K** represents the key type and **V** represents the type of values. For example,

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

Basic Operations on Java HashMap

The HashMap class provides various methods to perform different operations on HashMap. Some commonly used arraylist operations :

•Add elements

•Access elements

•Change elements

•Remove elements

1. ***Add elements to a HashMap***

- To add a element to the HashMap, we use the put() method of the HashMap class.

2. ***Access HashMap Elements***

- get() method is used to access the value from the HashMap.

3. ***Remove HashMap Elements***

- To remove elements from a HashMap, we can use the remove() method.

4. ***Change HashMap Elements***

- replace() method is used to change the value associated with a key in a HashMap.

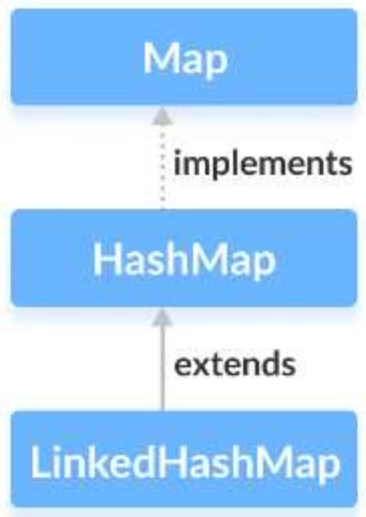
- ***Iterate through a HashMap***
- To iterate through each entry of the HashMap, we can use java for-each loop. We can iterate through **keys only**, **values only**, and **key/value mapping**.

Other Methods of HashMap

Method	Description
clear()	removes all mappings from the HashMap
merge()	merges the specified mapping to the HashMap
clone()	makes the copy of the HashMap
containsKey()	checks if the specified key is present in Hashmap
containsValue()	checks if Hashmap contains the specified value
size()	returns the number of items in HashMap
isEmpty()	checks if the Hashmap is empty

LinkedHashMap

- LinkedHashMap is concrete implementing class of map interface
- The LinkedHashMap interface extends the HashMap class to store its entries in a hash table. It internally maintains a doubly-linked list among all of its entries to order its entries
- Data stored in the form of key==value pair
- Insertion Order is maintained.
- It may have one null key and multiple null values.
- It contains unique elements.



Creating a LinkedHashMap

In order to create a linked HashMap, we must import the java.util.LinkedHashMap package first. Once we import the create LinkedHashMap in Java.

```
LinkedHashMap<Key, Value> map = new LinkedHashMap();
```

LinkedHashMap is created named map. Here,

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

1.Insert Elements to LinkedHashMap

- put() - inserts the specified key/value mapping to the map
- putAll() - inserts all the entries from the specified map to this map
- putIfAbsent() - inserts the specified key/value mapping to the map if the specified key is not present in the map

2.Access LinkedHashMap Elements

1. Using entrySet(), keySet() ,values() and get()

- entrySet() - returns a set of all the key/value mapping of the map
- keySet() - returns a set of all the keys of the map
- values() - returns a set of all the values of the map
- get() - returns the value associated with the specified key. If the key is not found, it returns null

3.Remove LinkedHashMap Elements

- remove(key) - returns and removes the entry associated with the specified key from the map
- remove(key, value) - removes the entry from the map only if the
specified key mapped to be the specified value and return a boolean value

Other Methods of LinkedHashMap

Method	Description
clear()	removes all the entries from the map
containsKey()	checks if the map contains the specified key and returns a boolean value
containsValue()	checks if the map contains the specified value and returns a boolean value
size()	returns the size of the map
isEmpty()	checks if the map is empty and returns a boolean value

LinkedHashMap Vs. HashMap

LinkedHashMap and HashMap implements the Map interface. However, there exist some differences between them.

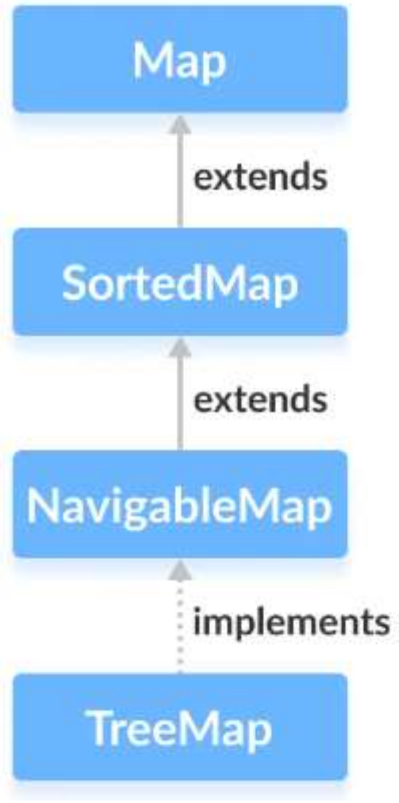
•LinkedHashMap maintains a doubly-linked list internally. Due to this, it maintains the insertion order of its elements.

•The LinkedHashMap class requires more storage than HashMap. This is because LinkedHashMap maintains linked lists internally.

•The performance of LinkedHashMap is slower than HashMap

TreeMap

- TreeMap is a concrete implementing class of Map interface
- The TreeMap class of the Java collections framework provides the tree data structure implementation.
- It implements the NavigableMap interface.
- It stores data in key==value pair
- It will sort the entries in the map with respect to keys in ascending order
- The key in TreeMap must be of comparable type. if it is not comparable type we get ClassCastException.
- In TreeMap key cannot be null, if it is null we get NullPointerException
- A value in TreeMap can be null.



- ***Creating a TreeMap.***

In order to create a TreeMap, we must import the java.util.TreeMap package first. Once the package is imported, we can create a TreeMap in Java

```
TreeMap<Key, Value> map = new TreeMap();
```

In the above code, TreeMap named map is created. The elements in TreeMap are sorted naturally (ascending order).

Here,

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

1.Insert Elements to TreeMap

- put() - inserts the specified key/value mapping (entry) to the map
- putAll() - inserts all the entries from specified map to this map
- putIfAbsent() - inserts the specified key/value mapping to the map if the specified key is not present in the map

2.Accessing TreeMap Elements

Using entrySet(), keySet() and values(),get()

•entrySet() - returns a set of all the key/values mapping (entry) of a treemap

•keySet() - returns a set of all the keys of a tree map

•values() - returns a set of all the maps of a tree map

•get() - Returns the value associated with the specified key. returns null if the key is not found.

3.Removing TeeMap Elements

•remove(key) - returns and removes the entry associated with the specified key from a TreeMap

•remove(key, value) - removes the entry from the map only if the specified key is associated with the specified value and returns a boolean value

Other Methods of TreeMap

Method	**Description**
clone()	Creates a copy of the TreeMap
containsKey()	Searches the TreeMap for the specified key and returns a boolean result
containsValue()	Searches the TreeMap for the specified value and returns a boolean result
size()	Returns the size of the TreeMap
clear()	Removes all the entries from the TreeMap

HashTable

- It is concrete implementing class of Map interface
- It is used to store data in key=value format
- In hashtable the insertion order is not maintained.
- In hashtable both key and value cannot be null. If it is null we get

  NullPointerException.
- The java.util.Hashtable class is a class in Java that provides a key-value data structure, similar to the Map interface.
- It was part of the original Java Collections framework and was introduced in Java 1.0.
- The Hashtable class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value
- ***Creating Hashtable*:**

```
Hashtable<K, V> ht = new Hashtable();
```

Example:

```
import java.util.Hashtable;

public class Main {
    public static void main(String[] args) {
        Hashtable<String, Integer> hashtable = new Hashtable<>();

        // Adding elements to the hashtable
        hashtable.put("A", 1);
        hashtable.put("B", 2);
        hashtable.put("C", 3);

        // Getting values from the hashtable
        int valueA = hashtable.get("A");
        System.out.println("Value of A: " + valueA);

        // Removing elements from the hashtable
        hashtable.remove("B");

            }
}
```