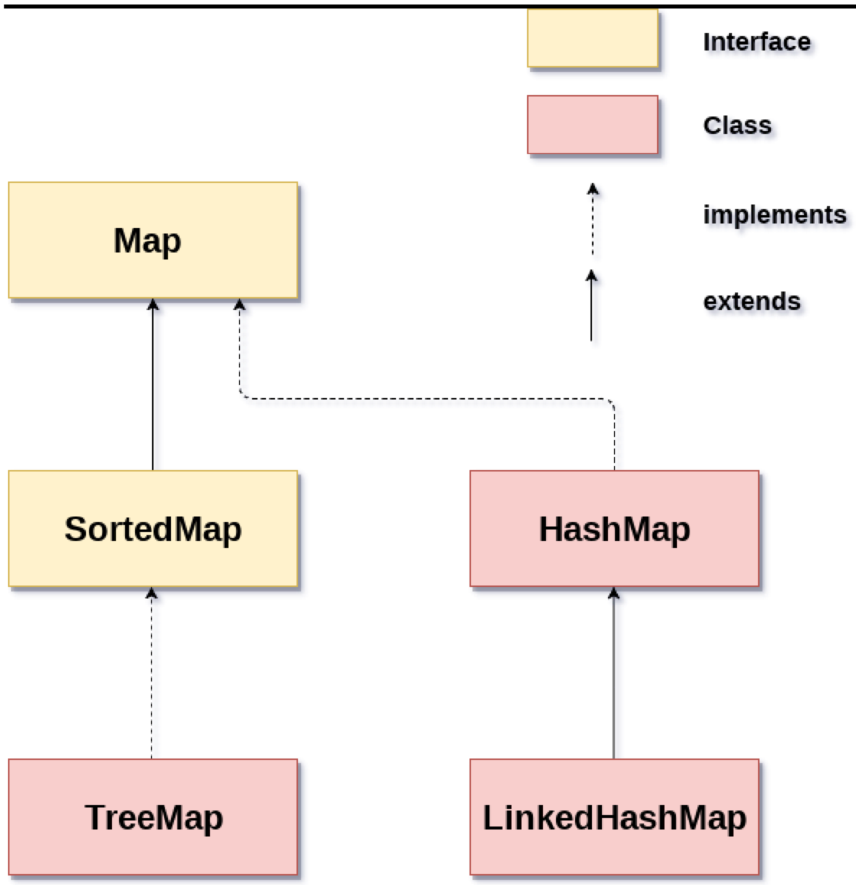
Q. What Is the Map interface in java?

Map interface is used for to store the data in the form of key and value pair. Key is unique identity of data and value is actual data present in map means we cannot add the duplicate key in map but we can add the duplicate value in map interface. In short we can say Map is combination of Set and List Collection Set represent key and List represent value.

Q. Explain the Map Hierarchy?

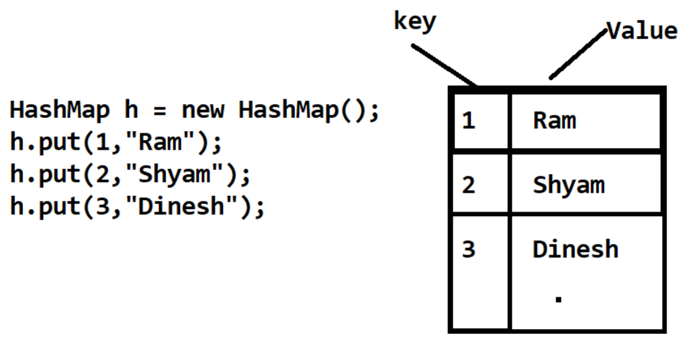


In the above Hierarchy we have the Map and SortedMap are the interfaces and HashMap and TreeMap are the implementer classes of above interfaces.

Q. Explain the Methods of Map interface

Map interface contain the some methods those help us to store and retrieve, delete, update etc operation on map interface
Some of the method given below with its example

void put(Object key, Object value): this method is used for to store data in map in the form of key and value pair.
Example

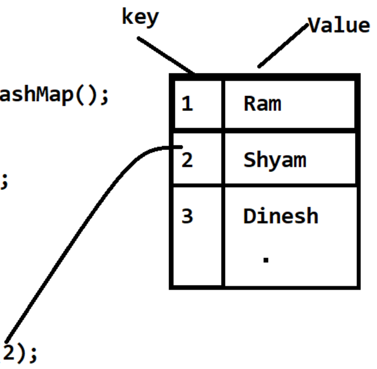


```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```

This method is used for store the data in map using key and value pair shown in diagram
Object get (Object key): this method is used for retrieve data from map using its key if key not found return null
Note: Normally we use this method for retrieve data using its key as well as this method help us to find or search data from map using its key shown in following example.

Example of Fetch Data from Map using its key

```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```

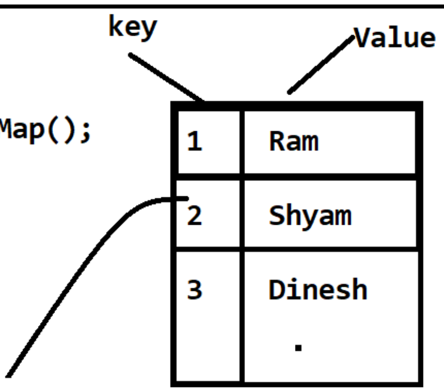


```
      Shyam
Object obj = h.get(2);

 System.out.println(obj); //Shyam
```

Note: in this example we pass the key get() method i.e 2 so we get its value Shyam

Example of Search Element from Map using get() method

```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```



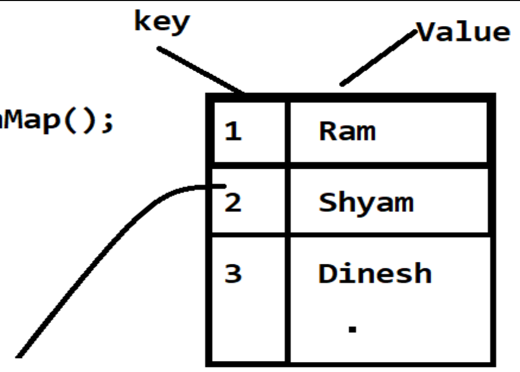
```
Object obj = h.get(2);

if(obj!=null)
{ System.out.println("Element found");
}
else{
 System.out.println("Element not found");
}
```

Note: get() method return null value if key not present in map and we check the condition obj!=null means if data found

boolean containsKey(Object key): this method check key present in map or not if present return true otherwise return false.

Example

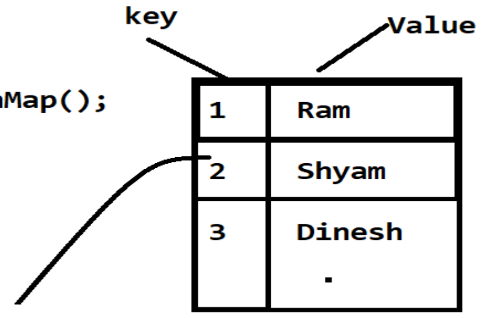
```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```



```
boolean  b= h.containsKey(2);
if(b)
{ System.out.println("Key present in map");
}
else{
  System.out.println("key not present in map");
 }
```

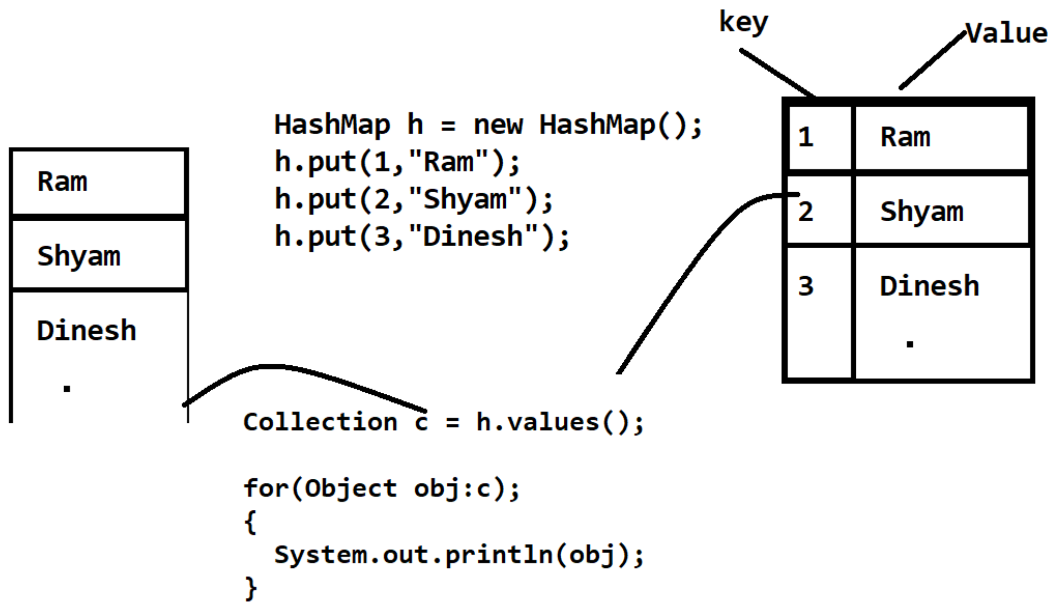
Object remove (Object key): this method can delete the data from Map using its key if key found the delete the element and return deleted value if key not found then return null value.

```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```



```
Object value= h.remove(2);

if(value!=null)
{
   System.out.println("Value deleted success");
}
else{
  System.out.println("Value not deleted");
}
```

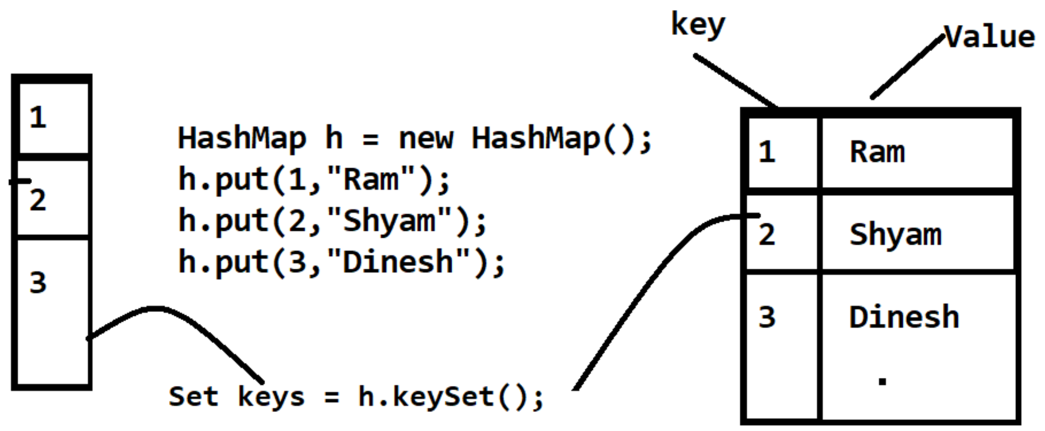
Collection values(): this method can return only values from Map interface



```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```

```
Collection c = h.values();

for(Object obj:c);
{
  System.out.println(obj);
}
```

Set keyset(): this method return the all keys from Map interface and store in Set Collection.

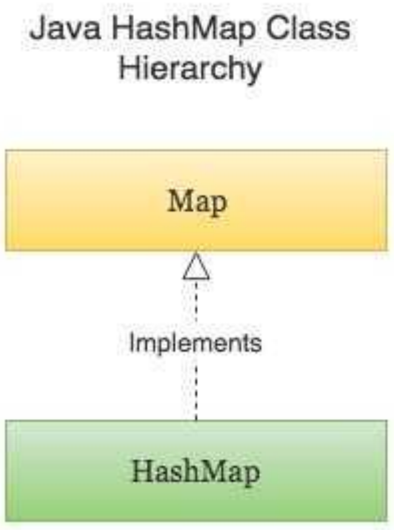


```
HashMap h = new HashMap();
h.put(1,"Ram");
h.put(2,"Shyam");
h.put(3,"Dinesh");
```

```
Set keys = h.keySet();
```

Q. Explain the HashMap?

Java HashMap is a hash table based implementation of Java’s Map interface. A Map, as you might know, is a collection of key-value pairs. It maps keys to values.



Following are few key points to note about HashMap in Java -

- A HashMap cannot contain duplicate keys.
- Java HashMap allows null values and the null key.
- HashMap is an unordered collection. It does not guarantee any specific order of the elements.
- Java HashMap is not thread-safe. You must explicitly synchronize concurrent modifications to the HashMap.

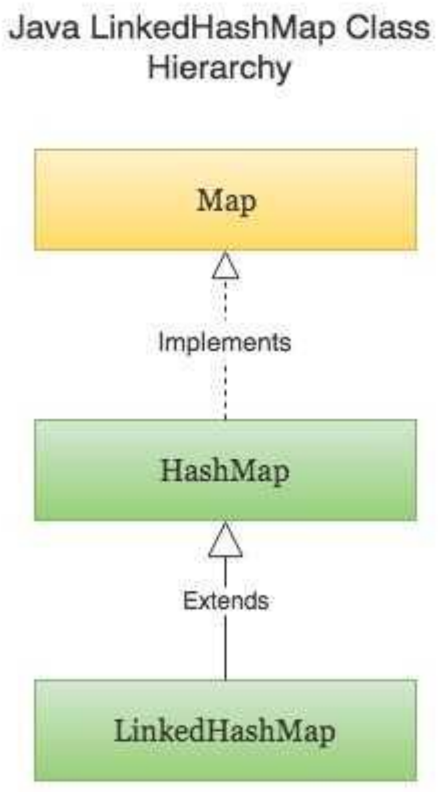
How to create the HashMap

```
import java.util.HashMap;
import java.util.Map;
public class CreateHashMapExample {
   public static void main(String[] args) {
      // Creating a HashMap
      Map numberMapping = new HashMap();
      // Adding key-value pairs to a HashMap
      numberMapping.put("One", 1);
      numberMapping.put("Two", 2);
      numberMapping.put("Three", 3);
```

```
        numberMapping.putIfAbsent("Four", 4);
        System.out.println(numberMapping);
    }
}
```

Q. Explain the LinkedHashMap In Java

Java LinkedHashMap is a hash table and doubly linked List based implementation of Java's Map interface. It extends the HashMap class which is another very commonly used implementation of the Map interface.



The iteration order in a LinkedHashMap is normally the order in which the elements are inserted. However, it also provides a special constructor using which you can change the iteration order from the least-recently accessed element to the most-recently accessed element and vice versa. This kind of iteration order can be useful in building LRU caches. In any case, the iteration order is predictable.

Following are some important points to note about LinkedHashMap in Java -

- A LinkedHashMap cannot contain duplicate keys.
- LinkedHashMap can have null values and the null key.
- Unlike HashMap, the iteration order of the elements in a LinkedHashMap is predictable.
- Just like HashMap, LinkedHashMap is not thread-safe. You must explicitly synchronize concurrent access to a LinkedHashMap in a multi-threaded environment.

Creating and Initializing a LinkedHashMap

```
import java.util.LinkedHashMap;
public class CreateLinkedHashMapExample {
   public static void main(String[] args) {
      // Creating a LinkedHashMap
      LinkedHashMap wordNumberMapping = new LinkedHashMap();

      // Adding new key-value pairs to the LinkedHashMap
      wordNumberMapping.put("one", 1);
      wordNumberMapping.put("two", 2);
      wordNumberMapping.put("three", 3);
      wordNumberMapping.put("four", 4);

         wordNumberMapping.putIfAbsent("five", 5);

      System.out.println(wordNumberMapping);
   }
}
```

Q. Perform Following Operation on LinkedHashMap

- Check if a key exists in a LinkedHashMap.
- Check if a value exists in the LinkedHashMap.
- Modify the value associated with a given key in the LinkedHashMap.

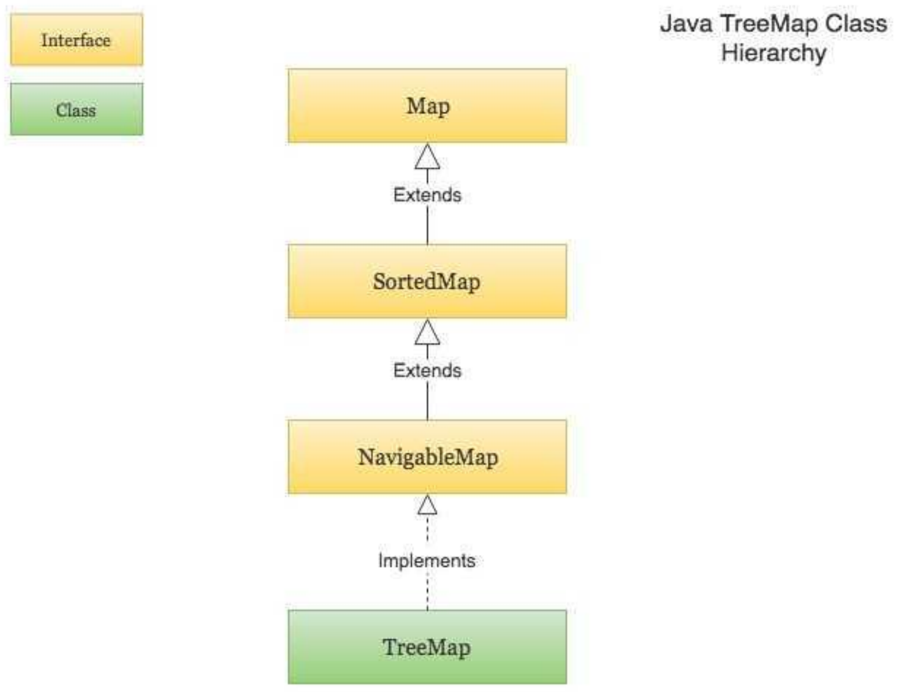
Example

```
import java.util.LinkedHashMap;
public class AccessEntriesFromLinkedHashMapExample {
   public static void main(String[] args) {
      LinkedHashMap<Integer, String> customerIdNameMapping = new
LinkedHashMap<>();
      customerIdNameMapping.put(1001, "Jack");
      customerIdNameMapping.put(1002, "David");
      customerIdNameMapping.put(1003, "Steve");
      customerIdNameMapping.put(1004, "Alice");
      customerIdNameMapping.put(1005, "Marie");
      System.out.println("customerIdNameMapping : " +
customerIdNameMapping);
        Integer id = 1005;
      if(customerIdNameMapping.containsKey(id)) {
         System.out.println("Found the customer with id " + id + " : " +
customerIdNameMapping.get(id));
      } else {
         System.out.println("Customer with id " + id + " does not exist");
      }
      String name = "David";
      if(customerIdNameMapping.containsValue(name)) {
         System.out.println("A customer named " + name + " exist in the map");
      } else {
     System.out.println("No customer found with name " + name + " in the map");
      }
```

```
 customerIdNameMapping.put(id, "Bob");
      System.out.println("Changed the name of customer with id " + id + ", New
mapping : " + customerIdNameMapping);
   }
}
```

Q. Explain the TreeMap in Detail

Java TreeMap is a Red-Black tree based implementation of Java's Map interface. The entries in a TreeMap are always sorted based on the natural ordering of the keys, or based on a custom Comparator that you can provide at the time of creation of the TreeMap.The TreeMap class is part of Java's collection framework. It implements the NavigableMap interface, which in turn extends the SortedMap interface. Following is the class hierarchy of TreeMap.



The SortedMap interface provides functionalities to maintain the ordering of keys. And the NavigableMap interface provides functionalities to navigate through the

map. For example, finding the entry just greater than or just less than the given key, finding the first and last entry in the TreeMap etc.
Since a TreeMap implements NavigableMap interface, it has the functionalities of both the NavigableMap as well as the SortedMap.

Following are few key points to note about TreeMap in Java -

- A TreeMap is always sorted based on keys. The sorting order follows the natural ordering of keys. You may also provide a custom Comparator to the TreeMap at the time of creation to let it sort the keys using the supplied Comparator.
- A TreeMap cannot contain duplicate keys.
- TreeMap cannot contain the null key. However, It can have null values.
- TreeMap is not synchronized. Access to TreeMaps must be synchronized explicitly in a multi-threaded environment.

Creating a TreeMap

```
import java.util.SortedMap;
import java.util.TreeMap;
public class CreateTreeMapExample {
   public static void main(String[] args) {
      // Creating a TreeMap
      SortedMap fileExtensions  = new TreeMap();
      // Adding new key-value pairs to a TreeMap
      fileExtensions.put("python", ".py");
      fileExtensions.put("c++", ".cpp");
      fileExtensions.put("kotlin", ".kt");
      fileExtensions.put("golang", ".go");
      fileExtensions.put("java", ".java");

      // Printing the TreeMap (Output will be sorted based on keys)
      System.out.println(fileExtensions);
```

```
        }
    }
```