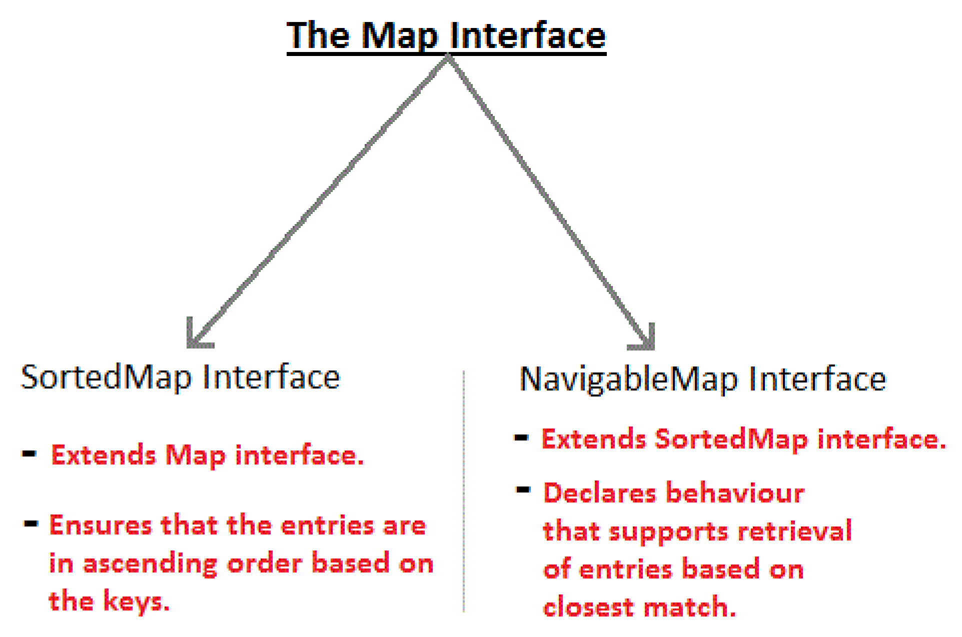
Map Interface - Java Collections

A Map stores data in key and value association. Both key and values are objects. The key must be unique but the values can be duplicate. Although Maps are a part of Collection Framework, they can not actually be called as collections because of some properties that they posses. However we can obtain a **collection-view** of maps.

It provides various classes: **HashMap, TreeMap, LinkedHashMap** for map implementation. All these classes implements Map interface to provide Map properties to the collection.

Map Interface and its Subinterface

Interface	Description
Map	Maps unique key to value.
Map.Entry	Describe an element in key and value pair in a map. Entry is sub interface of Map.
NavigableMap	Extends SortedMap to handle the retrienal of entries based on closest match searches
SortedMap	Extends Map so that key are maintained in an ascending order.


The Map Interface
SortedMap Interface
- Extends Map interface.
- Ensures that the entries are in ascending order based on the keys.
NavigableMap Interface
- Extends SortedMap interface.
- Declares behaviour that supports retrieval of entries based on closest match.

Map Interface Methods

These are commonly used methods defined by Map interface

- boolean **containsKey**(Object *k*): returns true if map contain *k* as key. Otherwise false.
- Object **get**(Object *k*) : returns values associated with the key *k*.
- Object **put**(Object *k*, Object *v*) : stores an entry in map.
- Object **putAll**(Map *m*) : put all entries from *m* in this map.

- Set **keySet**() : returns **Set** that contains the key in a map.
- Set **entrySet**() : returns **Set** that contains the entries in a map.

HashMap class

1. HashMap class extends **AbstractMap** and implements **Map** interface.
2. It uses a **hashtable** to store the map. This allows the execution time of get() and put() to remain same.
3. HashMap does not maintain order of its element.

HashMap has four constructor.

```
HashMap()

HashMap(Map< ? extends k, ? extends V> m)

HashMap(int capacity)

HashMap(int capacity, float fillratio)
```

Copy

HashMap Example

Lets take an example to create hashmap and store values in key and value pair. Notice to insert elements, we used put() method because map uses put to insert element, not add() method that we used in list interface.

```
import java.util.*;


class Demo

{

  public static void main(String args[])

  {

    HashMap< String,Integer> hm = new HashMap< String,Integer>();

    hm.put("a",100);

    hm.put("b",200);

    hm.put("c",300);

    hm.put("d",400);


    Set<Map.Entry<String,Integer>> st = hm.entrySet();   //returns Set
view

    for(Map.Entry<String,Integer> me:st)
    {
      System.out.print(me.getKey()+":");
      System.out.println(me.getValue());
    }
  }
}
```

Copy

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

TreeMap class

1. TreeMap class extends **AbstractMap** and implements **NavigableMap** interface.
2. It creates Map, stored in a tree structure.
3. A **TreeMap** provides an efficient means of storing key/value pair in efficient order.
4. It provides key/value pairs in sorted order and allows rapid retrieval.

Example:

In this example, we are creating treemap to store data. It uses tree to store data and data is always in sorted order. See the below example.

```
import java.util.*;

class Demo
{
  public static void main(String args[])
  {
    TreeMap<String,Integer> tm = new TreeMap<String,Integer>();
    tm.put("a",100);
    tm.put("b",200);
    tm.put("c",300);
    tm.put("d",400);

    Set<Map.Entry<String,Integer>> st = tm.entrySet();
    for(Map.Entry<String,Integer> me:st)

    {

      System.out.print(me.getKey()+":");

      System.out.println(me.getValue());

    }

  }

}
```

Copy

```
a:100

b:200

c:300

d:400
```

LinkedHashMap class

1. **LinkedHashMap** extends **HashMap** class.
2. It maintains a linked list of entries in map in order in which they are inserted.
3. **LinkedHashMap** defines the following constructor

```
4. LinkedHashMap()

5.

6. LinkedHashMap(Map< ? extends k, ? extends V> m)

7.

8. LinkedHashMap(int capacity)

9.

10.LinkedHashMap(int capacity, float fillratio)

11.
```

```
LinkedHashMap(int capacity, float fillratio, boolean order)
```

Copy

12. It adds one new method `removeEldestEntry()`. This method is called by `put()` and `putAll()` By default this method does nothing.

Example:

Here we are using linkedhashmap to store data. It stores data into insertion order and use linked-list internally. See the below example.

```java
import java.util.*;



class Demo

{

  public static void main(String args[])

  {

    LinkedHashMap<String,Integer> tm = new
LinkedHashMap<String,Integer>();

    tm.put("a",100);

    tm.put("b",200);

    tm.put("c",300);

    tm.put("d",400);



    Set<Map.Entry<String,Integer>> st = tm.entrySet();

    for(Map.Entry<String,Integer> me:st)

    {

      System.out.print(me.getKey()+":");

      System.out.println(me.getValue());

    }

  }

}
```

Copy

```
OUTPUT

a:100

b:200

c:300

d:400
```