Experiment No.15

Title: Implementing the concept of Map Classes
Aim: To study different Map Classes in java.

Theory:
A *map* is an object that stores associations between keys and values, or *key/value pairs*.

Given a key, you can find its value. Both keys and values are objects. The keys must be unique, but the values may be duplicated. Some maps can accept a **null** key and **null** values, others cannot.

Several classes provide implementations of the map interfaces. The classes that can be used for maps are summarized here:

Class	**Description**
AbstractMap	Implements most of the **Map** interface.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.

The HashMap Class:
The **HashMap** class uses a hash table to implement the **Map** interface. This allows the execution time of basic operations, such as **get( )** and **put( )**, to remain constant even for large sets.

The following constructors are defined:
HashMap( )
HashMap(Map *m*)
HashMap(int *capacity*)
HashMap(int *capacity*, float *fillRatio*)

The first form constructs a default hash map. The second form initializes the hash map by using the elements of *m*. The third form initializes the capacity of the hash map to *capacity*. The fourth form

initializes both the capacity and fill ratio of the hash map by using its arguments. The meaning of capacity and fill ratio is the same as for **HashSet**, described earlier.

HashMap implements **Map** and extends **AbstractMap**. It does not add any methods of its own

You should note that a hash map does *not* guarantee the order of its elements.
Therefore, the order in which elements are added to a hash map is not necessarily the order in which they are read by an iterator. The following program illustrates **HashMap**. It maps names to account balances.

Sample Program:

```
import java.util.*;
class HashMapDemo {
public static void main(String args[]) {
// Create a hash map
HashMap hm = new HashMap();
// Put elements to the map
hm.put("John Doe", new Double(3434.34));
hm.put("Tom Smith", new Double(123.22));
hm.put("Jane Baker", new Double(1378.00));
hm.put("Todd Hall", new Double(99.22));
hm.put("Ralph Smith", new Double(-19.08));
// Get a set of the entries
Set set = hm.entrySet();
// Get an iterator
Iterator i = set.iterator();
// Display elements
while(i.hasNext()) {
Map.Entry me = (Map.Entry)i.next();
System.out.print(me.getKey() + ": ");
System.out.println(me.getValue());
}
System.out.println();
// Deposit 1000 into John Doe's account
double balance = ((Double)hm.get("John Doe")).doubleValue();
hm.put("John Doe", new Double(balance + 1000));
System.out.println("John Doe's new balance: " +
hm.get("John Doe"));
}
}
```

The LinkedHashMap Class:
Java 2, version 1.4 adds the **LinkedHashMap** class. This class extends **HashMap**. **LinkedHashMap** maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map. That is, when iterating a **LinkedHashMap**, the elements will be returned in the order in which they were inserted. You can also create a **LinkedHashMap** that returns its elements in the order in which they were last accessed.

LinkedHashMap defines the following constructors.

LinkedHashMap( )
LinkedHashMap(Map *m*)
LinkedHashMap(int *capacity*)
LinkedHashMap(int *capacity*, float *fillRatio*)
LinkedHashMap(int *capacity*, float *fillRatio*, boolean *Order*)

The first form constructs a default **LinkedHashMap**. The second form initializes the **LinkedHashMap** with the elements from *m*. The third form initializes the capacity.
The fourth form initializes both capacity and fill ratio. The meaning of capacity and fill ratio are the same as for **HashMap**. The last form allows you to specify whether the elements will be stored in the linked list by insertion order, or by order of last access. If *Order* is **true**, then access order is used. If *Order* is **false**, then insertion order is used.

LinkedHashMap adds only one method to those defined by **HashMap**.
This method is **removeEldestEntry( )** and it is shown here.

protected boolean removeEldestEntry(Map.Entry *e*)

This method is called by **put( )** and **putAll( )**. The oldest entry is passed in *e*. By default, this method returns **false** and does nothing. However, if you override this method, then you can have the **LinkedHashMap** remove the oldest entry in the map. To do this, have your override return **true**.

Statement:

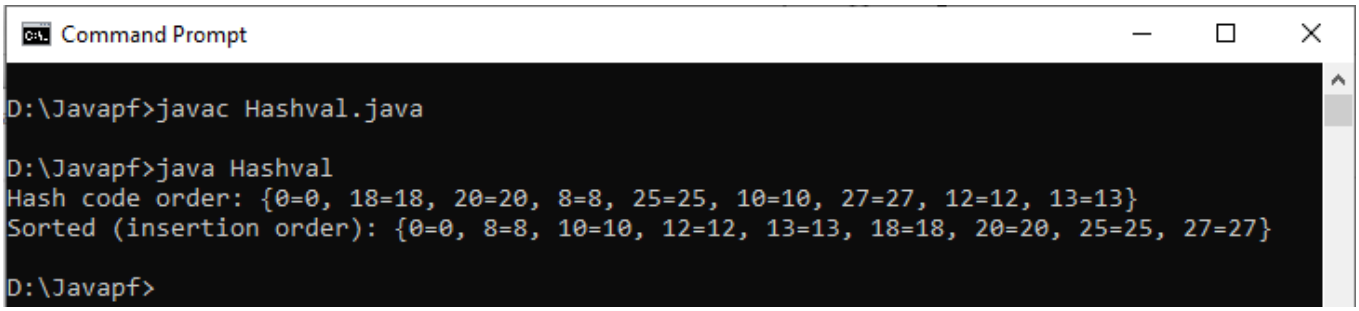
Fill a HashMap with key-value pairs. Print the results to show ordering by hash code. Extract the pairs, sort by key, and display the result.

Program:

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.Random;
import java.util.Set;

class HashMapper
{
    HashMap<Integer, String> map = new HashMap<Integer, String>();
    LinkedHashMap<Integer, String> linkedmap = new LinkedHashMap<Integer, String>();

    public void fillMap()
        {
        Random rand = new Random(42);
        int k;
        for (int i=0; i<10; i++)

{
            k = rand.nextInt(i+20);
            map.put(k, Integer.toString(k));
        }
        System.out.println("Hash code order: " + map);
    }

    public void remap()
        {
        Set<Integer> keyset = map.keySet();
        Iterator<Integer> it;
        int  temp;
        int smallest;
        int iterations = keyset.size();
        for (int i = 0; i < iterations; i++)
        {
            it = keyset.iterator();
            smallest = it.next();
            it = keyset.iterator();
            while(it.hasNext())
            {
```

```
                    temp = it.next();
                    if (temp < smallest) smallest = temp;
                }
                linkedmap.put(smallest, map.get(smallest));
                keyset.remove(smallest);
            }
            System.out.println("Sorted (insertion order): " + linkedmap);
        }
}


public class Hashval
{
    public static void main(String[] args)
    {
            HashMapper hm = new HashMapper();
            hm.fillMap();
            hm.remap();
    }
}
```

Output:-



Conclusion: hence we have studied different Map Classes and implement it.