Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No. 08

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : HashMap and HashTable

AIM: In an array 1-100 many numbers are duplicates. Use hash Map. Given two arrays 1,2,3,4,5 and 2,3,1,0,5. Find which element is not present in the second array. Use hash Table.

Expected OUTCOME of Experiment:

CO2: Solve problems using Java basic constructs (like if else statement, control structures, and data types, array, string, vectors, packages, collection class).

Books/ Journals/ Websites referred:

1.Ralph Bravaco , Shai Simoson , "Java Programing From the Group Up" Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design .

Pre Lab/ Prior Concepts:

Hashtable:

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.
To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

- It is similar to HashMap, but is synchronised.
- Hashtable stores key/value pair in hash table.
- In Hashtable we specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

Constructors:

- **Hashtable():** This is the default constructor.

- **Hashtable(int size):** This creates a hash table that has initial size specified by size.
- **Hashtable(int size, float fillRatio):** This version creates a hash table that has initial size specified by size and fill ratio specified by fillRatio. **fill ratio:** Basically it determines how full hash table can be before it is resized upward.and its Value lie between 0.0 to 1.0
- **Hashtable(Map m):** This creates a hash table that is initialised with the elements in m.

Methods:

1. **void clear() :** method clears the hashtable so that it contains no keys.
2. **Object clone() :** used to create a shallow copy of this hashtable.

```
// Java code illustrating clear() and clone() methods
import java.util.*;
class hashTabledemo
{
    public static void main(String[] arg)
    {
          //creating a hash table
        Hashtable h = new Hashtable();

        Hashtable h1 = new Hashtable();

        h.put(3, "Geeks");
        h.put(2, "forGeeks");
        h.put(1, "isBest");

        // create a clone or shallow copy of hash table h
        h1 = (Hashtable)h.clone();

        // checking clone h1
        System.out.println("values in clone: " + h1);

        // clear hash table h
        h.clear();

        // checking hash table h
        System.out.println("after clearing: " + h);
      }
}
```

HashMap:

HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a large String to small

String that represents same String. A shorter value helps in indexing and faster searches. HashSet also uses HashMap internally. It internally uses a link list to store key-value pairs already explained in HashSet in detail and further articles.
Few important features of HashMap are:

- HashMap is a part of java.util package.
- HashMap extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface.
- It also implements Cloneable and Serializable interface. K and V in the above definition represent Key and Value respectively.
- HashMap doesn't allow duplicate keys but allows duplicate values. That means A single key can't contain more than 1 value but more than 1 key can contain a single value.
- HashMap allows null key also but only once and multiple null values.
- This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time. It is roughly similar to HashTable but is unsynchronized.

Internal Structure of HashMap

Internally HashMap contains an array of Node and a node is represented as a class which contains 4 fields :

1. int hash
2. K key
3. V value
4. Node next

Constructors in HashMap

HashMap provides 4 constructors and access modifier of each is public:

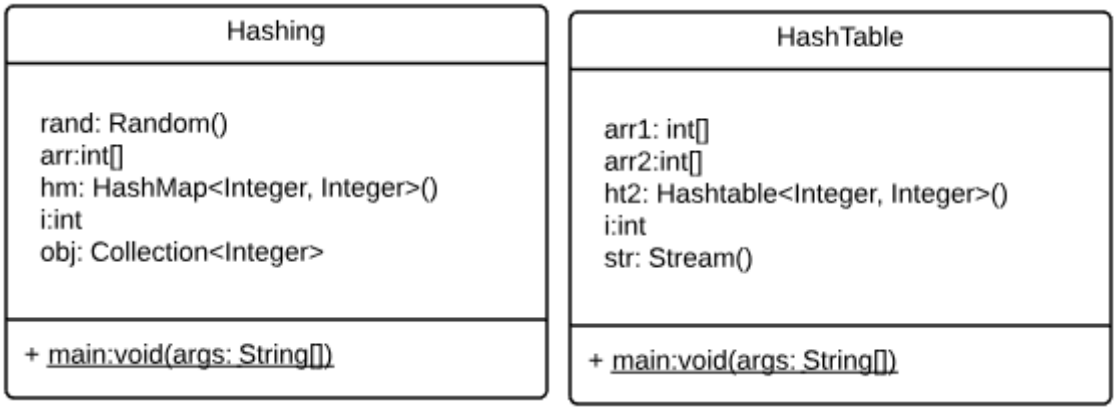
1. **HashMap() :** It is the default constructor which creates an instance of HashMap with initial capacity 16 and load factor 0.75.
2. **HashMap(int initial capacity) :** It creates a HashMap instance with specified initial capacity and load factor 0.75.
3. **HashMap(int initial capacity, float loadFactor) :** It creates a HashMap instance with specified initial capacity and specified load factor.
4. **HashMap(Map map) :** It creates instance of HashMapwith same mappings as specified map.

Methods in HashMap

1. void clear(): Used to remove all mappings from a map.
2. boolean containsKey(Object key): Used to return True if for a specified key, mapping is present in the map.
3. boolean containsValue(Object value): Used to return true if one or more key is mapped to a specified value.
4. Object clone(): It is used to return a shallow copy of the mentioned hash map.
5. boolean isEmpty(): Used to check whether the map is empty or not. Returns true if the map is empty.
6. Set entrySet(): It is used to return a set view of the hash map.

7. Object get(Object key): It is used to retrieve or fetch the value mapped by a particular key.
8. Set keySet(): It is used to return a set view of the keys.
9. int size(): It is used to return the size of a map.
10. Object put(Object key, Object value): It is used to insert a particular mapping of key-value pair into a map.
11. putAll(Map M): It is used to copy all of the elements from one map into another.
12. Object remove(Object key): It is used to remove the values for any particular key in the Map.
13. Collection values(): It is used to return a Collection view of the values in the HashMap.

Class Diagram:



Algorithm:

STEP 1: START.

STEP 2: Instantiate a HashMap object and create an array of random numbers.

STEP 3: Run a loop to iterate through the array and insert the elements in the HashMap with the random number, i.e. array element as key and its count as value. If the HashMap already contains the key, then its value, i.e. count is incremented by 1.

STEP 4: Print the HashMap object that will display the elements and number of occurrences for each of them.

STEP 5: Instantiate two arrays of given elements and a HashTable that stores the elements of second array in it.

STEP 6: Run a loop to iterate through the first array and check whether the current element of first array is present in the HashTable object. If not, print it.

STEP 7: END.

Implementation details:

HashMap:

```
import java.util.*;

class Hashing{
  public static void main(String[] args){
    Random rand=new Random();
    int arr[]=new int[15];
    HashMap<Integer, Integer> hm=new HashMap<Integer, Integer>();
    for(int i=0;i<15;i++){
        arr[i]=rand.nextInt(10);
        hm.put(i,arr[i]);
    }
    System.out.println("HashMap before removing duplicates: ");
     for (Map.Entry<Integer, Integer> entry : hm.entrySet())
            System.out.println("Key = " + entry.getKey() +", Value = " +
entry.getValue());
        System.out.println();
    Collection<Integer> obj=hm.values();
    for(Iterator<Integer> itr=obj.iterator();itr.hasNext();){
      if(Collections.frequency(obj, itr.next())>1)
        itr.remove();
    }
    System.out.println("HashMap after removing duplicates: ");
     for (Map.Entry<Integer, Integer> entry : hm.entrySet())
            System.out.println("Key = " + entry.getKey() +", Value = " +
entry.getValue());
    }
}
```

HashTable:

```
import java.util.*;
class HashTable{
  public static void main(String[] args){
    int arr1[]={1,2,3,4,5};
    int arr2[]={2,3,1,0,5};
    Hashtable<Integer, Integer> ht2=new Hashtable<Integer, Integer>();
```

```
    for(int i=0;i<arr2.length;i++)
      ht2.put(i,arr2[i]);
    System.out.println("Array A: ");
    Arrays.stream(arr1).forEach(str -> System.out.print(str + " "));
    System.out.println("\nArray B: ");
    Arrays.stream(arr2).forEach(str -> System.out.print(str + " "));
    System.out.print("\nThe elements from A not present in B are: ");
    for(int i=0;i<arr1.length;i++){
      if(!ht2.containsValue(arr1[i]))
        System.out.print(arr1[i]+" ");
    }
  }
}
```

My code can be found at:
https://repl.it/@ARGHYADEEPDAS/HashTableAndHashMap

Output Screen:

```
C:\Users\OAD\Desktop\1711072>java HashTable
Array A:
1 2 3 4 5
Array B:
2 3 1 0 5
The elements from A not present in B are: 4
```

```
C:\Users\OAD\Desktop\1711072>java Hashing
HashMap before removing duplicates:
Key = 0, Value = 6
Key = 1, Value = 7
Key = 2, Value = 1
Key = 3, Value = 2
Key = 4, Value = 7
Key = 5, Value = 2
Key = 6, Value = 1
Key = 7, Value = 6
Key = 8, Value = 3
Key = 9, Value = 4
Key = 10, Value = 1
Key = 11, Value = 6
Key = 12, Value = 8
Key = 13, Value = 3
Key = 14, Value = 9

HashMap after removing duplicates:
Key = 4, Value = 7
Key = 5, Value = 2
Key = 9, Value = 4
Key = 10, Value = 1
Key = 11, Value = 6
Key = 12, Value = 8
Key = 13, Value = 3
Key = 14, Value = 9
```

Conclusion: The program ran successfully as we were able to utilize HashMap and HashTable correctly. In addition, we were able to explore Collection and Collections.

Date: 02/11/2018 **Signature of faculty in-charge**