L58

Monday, February 7, 2022 4:27 PM

Data Structures - hashCode(), Map, HashMap, TreeMap, LinkedHashMap

Saurav Hathi

https://www.youtube.com/channel/UCp6MFWao5vWRnyRCxBsKnfw

https://www.instagram.com/saurav_hathi/

Understanding hashCode()

Q1. The Object class defines a method called hashCode() which returns an int value.

This int value is called hashcode value of that object.

This int value is called hashcode because it is used in hash table implementations like HashMap, Hashtable etc.

Hash tables are data structures that map **keys** to **values**.

Simple arrays and all subtypes of Collection interface like List, Set and Queue are called linear data structures, because they store only elements.
For example: ["A", "B", "C"] is a example of a linear structure.

A Hash table, map, associative array, dictionary are data structures which store a collection of **key and value pairs** (Key Value).
For example:[("IN" "INDIA"), ("US" "United States of America"), ("UK" "United Kingdom")]

In Java, the Map interface provides the functionality of a hash table or associative array, where keys are mapped to values.

HashMap is an implementation of Map (we will learn more about **Map** later)

A HashMap is a Map implementation which internally uses hashing function to enable extremely fast storage and retrieval.

In this context the hashcode value returned by the hashCode() method of the Object class is used while storing the **key and value** pairs.

Since the hashCode() method is present in the Object class, every class in Java inherits the default implementation. Classes can also provide a custom implementation as long as they follow the below simple rules:

1. The hashCode() method must return an int value.
2. The hashCode() method when invoked multiple times on the same object should return the same value.
3. The above rule is applicable during a single run of the Java program (JVM). Meaning the value returned by the method can differ during different runs of the Java application (JVM).
4. If two objects are equal according to equal() method, then hashCode() method on both those objects must also return same integer value.
5. If two objects are not equal according to equal() method, then hashCode() method need not return different values. However, returning different values for unequal objects increases the performance of HashMaps.

Note that whenever equals(Object obj) method of Object is overridden in a subclass, the hashcode() method also should be overridden.

See and retype the below code.

Generally in our day-to-day programming we do not call the hashCode() method directly. As stated above it is internally used by **HashMap** and **Hashtable** while storing an object as a key.

```java
package q11370;
public class HashCodeDemo {
    public static void main(String[] args) {
        String text1 = "Ganga";
        System.out.println("text1 = " + text1 + " text1.hashCode() = " + text1.hashCode());
        String text2 = "GangaRiver";
        System.out.println("text2 = " + text2 + " text2.hashCode() = " + text2.hashCode());
        String text3 = text2.substring(0, 5);
        System.out.println("text3 = " + text3 + " text3.hashCode() = " + text3.hashCode());
    }
}

```

Understanding Map Interface

Q1. A Map represents a collection of key-value pairs.

In a Map one key is mapped to at most one value.

Map does not allow duplicate keys, but the values can be duplicated.

Below are some of the concrete classes which implement Map interface:

1. HashMap - most commonly used class whenever we want to store key mapped with values (the order of iteration over the entries is not guaranteed to be the same as the insertion order).
2. TreeMap - is used when we want the key-value entries to be sorted on the natural ordering of its keys, or by a custom Comparator
3. LinkedHashMap - is used when we want the retrieval of key-value entries to be in the order in which they were inserted or in the order in which they were last accessed.

The most commonly used methods in Map interface are given below:

1. **put**(K key, V value) - stores the key and value mapping. If the key already exists, the old value will be replaced with the newly provided value and the old value will be returned.
2. **get**(Object key) - returns the value mapped to the given key.
3. **remove**(Object key) - removes the key-value mapping if such a mapping exists and returns the value mapped to the key. It returns null, if there is no mapping for the given key.
4. **size**() - returns the count of key-value pairs present.
5. **keySet**() - returns a Set containing only the keys in this map.
6. **putAll**(Map m) - stores all the the key and value mapping in the given map **m**.

Complete the missing code below.

```java
package q11371;
import java.util.*;
public class MapDemo
{
  public static void main (String[]args)
  {
      Map countryCodesMap = new HashMap ();
      countryCodesMap.put ("IN", "India");

    /*Follow given instructions
       see how key and values is added to the map similarly add the following data

       put key US and value as United States of America

       put key UK and value as United Kingdom
     */
      countryCodesMap.put ("US", "United States of America");
      countryCodesMap.put ("UK", "United Kingdom");

      System.out.println ("Map entries : " + countryCodesMap);

      countryCodesMap.put ("IN", "Bharat");
    // countryCodesMap.put("UK","United Kingdom");

      System.out.println ("Map entries : " + countryCodesMap);
      System.out.println ("Value for UK is : " + countryCodesMap.get ("UK"));

    //Fill the missing code here get the value for key UK
      System.out.println ("Removing entry for US : " +countryCodesMap.remove ("US"));

    //Fill the missing code remove the key US
      System.out.println ("Map entries : " + countryCodesMap);
  }
}
```

HashMap

Q1. Whenever we want to store large amount of data (such that each date item can be uniquely identified by an id or a key) and also be able to retrieve the data quickly, we use a HashMap.

HashMap has 4 constructors.

1. HashMap() - the default constructor creates an empty HashMap with initialCapacity as 16 and a default load factor of 0.75
2. HashMap(int initialCapacity) - it creates an empty HashMap with the given initial capacity and a default load factor of 0.75.
3. HashMap(int initialCapacity, float loadFactor) - it creates an empty HashMap with the given initial capacity and load factor.
4. HashMap(Map m) - it creates a HashMap with the key-value mappings present in the map m passed as parameter.

HashMap internally creates an entry object for every key and value mapping.

These entry objects are placed in buckets/slots. **Capacity** is number of such slots/buckets. The capacity at the time of creation of a **HashMap** is called **initialCapacity**.

Note that **size** of the **HashMap** is different from the **capacity**. **Size** is the total number of entries inserted into the **HashMap**.

The **load factor** determines at what level of fullness the HashMap's capacity should be automatically increased.

The increase in the capacity is performed when the size becomes greater than (load factor x current capacity).

During the increase in capacity, the HashMap internally performs rehashing of the keys to store them in the new slots/buckets. This is where the **hashcode** of the keys is used by the **HashMap**.

Note that whenever we call the size() method on an HashMap, it always returns the current count of key and value entries it holds.

See and retype the below code.
You will notice that the size of cMap is 0, even though we create it with an **initialCapacity** of 20. This is because we have not added any entries to cMap.

When we know the count of key-value pairs we will be storing in an HashMap (assuming they are greater than **16**), it is efficient to provide it as the initialCapacity so that the HashMap can avoid frequent internal capacity adjustments during insertions.

```java
package q11372;
import java.util.*;
public class HashMapDemo {
    public static void main(String[] args) {
        Map aMap = new HashMap();
        System.out.println("aMap.size() = " + aMap.size());
        System.out.println("aMap = " + aMap);
        aMap.put("1", "First Entry");
        aMap.put("2", "Second Entry");
        aMap.put("3", "Third Entry");
        aMap.put("4", "Fourth Entry");
        System.out.println("aMap.size() = " + aMap.size());
        System.out.println("aMap = " + aMap);
        Map bMap = new HashMap(aMap);
        System.out.println("bMap.size() = " + bMap.size());
        System.out.println("bMap = " + bMap);
        Map cMap = new HashMap(20);
        System.out.println("cMap.size() = " + cMap.size());
        System.out.println("cMap = " + cMap);
    }
}
```

Q2. See the code and retype the same to learn how to iterate over the entries stored in a HashMap.

Note the usage of HashMap class and the iterator method.

The class scans through all the arguments passed to the mainmethod, and stores them into a HashMap with the argument's **first three chars** as key and the argument as the value.

We can assume the size of names passed as arguments will be greater than three characters.

The code uses the keySet() method in HashMap, which returns all the keys in a Set.

The program uses the **for-each** loop to iterate on the Set of keys, to print all keys along with their associated values.

Note that the keys which are retrieved are not in the order of the elements passed into the main method.

```
package q11373;
import java.util.*;
public class HashMapIterationDemo {
    public static void main(String[] args) {
        Map namesMap = new HashMap();
        for (String argument : args) {
            String shortName = argument.substring(0, 3);
            namesMap.put(shortName, argument);
        }
        Set shortNamesSet = namesMap.keySet();
        for (Object key : shortNamesSet) {
            Object value = namesMap.get(key);
            System.out.println(key + " : " + value);
        }
    }
}

```

Q3. See the code and retype the same to learn how to iterate over the entries stored in a HashMap using entrySet() method.

The class scans through all the arguments passed to the mainmethod, and stores them into a HashMap with the argument's **first three chars** as key and the argument as the value.

We can assume the size of names passed as arguments will be greater than three characters.

The code uses the entrySet() method in HashMap, which returns all the entries in a Set. The entries are objects of class Map.Entry interface.

The program uses the **for-each** loop to iterate on the Set of entries, to print key and value stored in each entry.

Also note how we are type-casting the entryObject which is of type java.lang.Object into Map.Entry, so that we can call the methods getKey() and getValue() which are present in Map.Entry interface.

Note that the keys which are retrieved are not in the order of the elements passed into the main method, this is because HashMap does not guarantee the order of of the entries.

```
package q11374;
import java.util.*;
public class HashMapIterationDemo {
    public static void main(String[] args) {
        Map namesMap = new HashMap();
        for (String argument : args) {
            String shortName = argument.substring(0, 3);
            namesMap.put(shortName, argument);
        }
        Set entrySet = namesMap.entrySet();
        for (Object entryObject : entrySet) {
            Map.Entry entry = (Map.Entry)entryObject;
            Object key = entry.getKey();
            Object value = entry.getValue();
            System.out.println(key + " : " + value);
        }
    }
}

```

Q4. A HashMap implementation provides a constant-time performance for the put and get methods.

However, the HashMap does not guarantee that the order of retrieval of entries will be same as its size grows.

Fill the missing code given below which illustrates some of the commonly used methods in HashMap.

Correlate the code and output to understand the usage of the methods put(K key, V value) and get(Object key).

```java
package q11375;
import java.util.*;
public class HashMapMethodsDemo
{
  public static void main (String[]args)
  {
    Map namesMap = new HashMap ();
      namesMap.put ("Sam", "Samos");
    /*
       Observe how key, value is added to map

       Similarly add the following (key, value) pairs to the map

       put (Hyd,Hyderabad )

       put (Dal, Dallas)

       put(Ban, Bangalore)

     */
      namesMap.put ("Hyd", "Hyderabad");

      namesMap.put ("Dal", "Dallas");

      namesMap.put ("Ban", "Bangalore");

      System.out.println ("namesMap = " + namesMap);

      System.out.println ("value mapped to Dal is : " + namesMap.get ("Dal"));  //get the value mapped to Dal

      namesMap.put ("Dal", "Dalton");

      System.out.println ("namesMap = " + namesMap);

      System.out.println ("new value mapped to Dal is : " +
              namesMap.get ("Dal"));

      System.out.println ("namesMap.size() = " + namesMap.size ());

  }
}

```

TreeMap

Q1. The TreeMap class implements NavigableMap. NavigableMap extends the SortedMap and the SortedMap in turn extends Map interface.

Unlike a HashMap, implementations of a SortedMap interface guarantee a sorted order (ascending) on the keys. The sort order can also be controlled by providing a custom Comparator implementation.

A NavigableMap interface extends SortedMap, and additionally provides navigation methods for navigating on the sorted entries.

TreeMap is a concrete implementation of SortedMap and NavigableMap interfaces.

See and retype the below code. You will notice that the entries in the TreeMap always remain sorted on the ascending order of their keys.

```java
package q11376;
import java.util.*;
public class TreeMapDemo {
    public static void main(String[] args) {
        Map namesMap = new TreeMap();
        namesMap.put("Sam", "Samos");
        namesMap.put("Hyd", "Hyderabad");
        namesMap.put("Dal", "Dallas");
        System.out.println("namesMap = " + namesMap);
        namesMap.put("Ban", "Bangalore");
        System.out.println("namesMap = " + namesMap);
        namesMap.put("Ath", "Athens");
        System.out.println("namesMap = " + namesMap);
    }
}

```

LinkedHashmap

Q1. The LinkedHashMap is a subclass of HashMap.

Unlike a HashMap which does not maintain order of the added entries, a LinkedHashMap orders the entries in their insertion order by default.

A LinkedHashMap also has a special constructor LinkedHashMap(int initialCapacity, float loadFactor, boolean accessOrder) which creates a map whose order of iteration will be the order in which its entries were last accessed, i.e the order will be from least-recently accessed to most-recently accessed.

Fill the missing code in the below program. You will notice that the entries in the LinkedHashMap always maintain their insertion order.

```java
package q11377;
import java.util.*;
public class LinkedHashMapDemo
{
  public static void main (String[]args)
  {
    Map namesMap = new LinkedHashMap ();
      namesMap.put ("Sam", "Samos");
    /*
      Add the following key value pair to the map

      add (Hyd, Hyderabad)

      add (Dal, Dallas)

    */

      namesMap.put ("Hyd", "Hyderabad");

      namesMap.put ("Dal", "Dallas");

      System.out.println ("namesMap = " + namesMap);

    //Now add (Ban, Bangalore)

      namesMap.put ("Ban", "Bangalore");

    //Print the data in namesMap

      System.out.println ("namesMap = " + namesMap);

    //add (Ath, Athens)

      namesMap.put ("Ath", "Athens");
      System.out.println ("namesMap = " + namesMap);
  }
}
```

Practice Programs on HashMap

Q1. A HashMap implementation provides a constant-time performance for the put and get methods.

put(K key, V value): Add the specified value with the specified key in the map. If the map previously contained a mapping for the key, the old value is replaced with the new one.

Write a program to understand how the (key, value) pair is inserted into HashMap using the method put.

Create a class HashMapMethodsDemo with a main method. Create an instance of the HashMap and add the given (key, values) into the map and print the result. The (key, values) are

- (Jan, January)
- (Feb, February)
- (Mar, March)
- (Apr, April)

The result should be as follows:
namesMap = {Feb=February, Apr=April, Jan=January, Mar=March}
namesMap = {Feb=Fabulous, Apr=April, Jan=January, Mar=March}

```java
package q11963;
import java.util.*;
public class HashMapMethodsDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new HashMap < String, String > ();

    // add given (key, values) to Map

    namesMap.put ("Jan", "January");

    namesMap.put ("Feb", "February");

    namesMap.put ("Mar", "March");

    namesMap.put ("Apr", "April");


    System.out.println ("namesMap = " + namesMap);

    // change the value February to Fabulous

    namesMap.put ("Feb", "Fabulous");
    System.out.println ("namesMap = " + namesMap);


  }
}
```

Q2. **get(Object key)**: Returns the value to which the key is mapped, or returns null if there is no mapping for the key.

Write a program to understand how to get the value mapped to the particular key in HashMap using get method.

Create a class HashMapMethodsDemo with a main method. Create an instance of the HashMap and get the value mapped to the key Tue.

The result should be as follows:
namesMap = {Thu=Thursday, Tue=Tuesday, Sun=Sunday, Mon=Monday}

value mapped to Tue is : Tuesday

```java
package q11968;
import java.util.*;
public class HashMapMethodsDemo {
    public static void main(String[] args) {
        Map <String, String> namesMap = new HashMap<String, String>();
        namesMap.put("Sun", "Sunday");
        namesMap.put("Mon", "Monday");
        namesMap.put("Tue", "Tuesday");
        namesMap.put("Thu", "Thursday");
        System.out.println("namesMap = " + namesMap);

        // write your code here

        System.out.println("value mapped to Tue is : " + namesMap.get("Tue"));

    }
}
```

Q3. Create a class HashMapIterationDemo with a main method. The method takes inputs from the command line arguments. From the input make the **first two chars** as key and the argument as value. Print all the (key, value) pairs. We can assume the size of names passed as arguments will be greaterthan three characters.

The code uses the keySet() method in HashMap, which returns all the keys in a Set.

Use **for-each** loop to iterate on the Set of keys, to print all keys along with their associated values.

The result should be as follows:
Cmd Args : Sunday Monday Tuesday Wednesday
Tu : Tuesday
Su : Sunday
Mo : Monday
We : Wednesday

```java
import java.util.*;
public class HashMapIterationDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new HashMap < String, String > ();
    Set shortNamesSet = namesMap.keySet ();

    for (int i = 0; i < args.length; i++)
      {

    String key = args[i].substring (0, 2);

    String value = args[i];

      namesMap.put (key, value);

      }




    Set < String > keys = namesMap.keySet ();

  for (Object key:shortNamesSet)
      {

    System.out.println (key + " : " + namesMap.get (key));
      }


  }
}
```

Q4. Create a class HashMapIterationDemo with a main method. The method takes inputs from the command line arguments. From the input make the **first two chars of the argument** as key and the entire argument as value. Print the result as shown in the example.

We can assume the size of names passed as arguments will be greater than three characters.

Sample Input and Output
Cmd Args : Red White Black Brown
{Br=Brown, Wh=White, Re=Red, Bl=Black}

```java
package q24086;
import java.util.*;
public class HashMapIterationDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new HashMap < String, String > ();
    Set shortNamesSet = namesMap.keySet ();
    /*for */

    for (int i = 0; i < args.length; i++)
      {

    String key = args[i].substring (0, 2);

    String value = args[i];

      namesMap.put (key, value);
      }


    System.out.println (namesMap);

  }
}

```

Q5. Create a class HashMapIterationDemo with a main method. The method takes inputs from the command line arguments. From the input make the **first character** as key and the entire argument as value. Print all the keys.

We can assume the size of names passed as arguments will be greater than three characters.

The code uses the keySet() method in HashMap, which returns all the keys in a Set.

Use **for-each** loop to iterate on the Set of keys, to print all keys.

The result should be as follows:
Cmd Args : Red Green Yellow Brown Black
{R=Red, B=Black, G=Green, Y=Yellow}
R
B
G
Y

```java
package q24088;
import java.util.*;
public class HashMapIterationDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new HashMap < String, String > ();

    for (int i = 0; i < args.length; i++)
      {

    // iterate over all the input argumetnts and add the (key,value) to the Map

    // write your code here

    String key = args[i].substring (0, 1);

    String value = args[i];

      namesMap.put (key, value);

      }
    System.out.println (namesMap);

    Set shortNamesSet = namesMap.keySet ();

    // get all the keys and print the result

    // write your code here

  for (Object key:shortNamesSet)
      {

    System.out.println (key);

      }
  }
}

```

Q6. Create a class HashMapIterationDemo with a main method. The method takes inputs from the command line arguments. From the input make the **first two chars of arguments** as key , and the total argument as value. Print all the values. We can assume the size of names passed as arguments will be greater than three characters.

The code uses the keySet() method in HashMap, which returns all the keys in a Set.

Use **for-each** loop to iterate on the Set of keys, to print all keys along with their associated values.

The result should be as follows:
Cmd Args : One Two Three Four
{Tw=Two, Th=Three, Fo=Four, On=One}
Two
Three
Four
One

```java
import java.util.*;
public class HashMapIterationDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new HashMap < String, String > ();
    Set shortNamesSet = namesMap.keySet ();
    for (int i = 0; i < args.length; i++)
      {

    String key = args[i].substring (0, 2);

    String value = args[i];

      namesMap.put (key, value);

      }

    System.out.println (namesMap);

  for (Object key:shortNamesSet)
      {
    // get all the values and print the result
    System.out.println (namesMap.get (key));

      }
  }
}

```

Q7. Create a class CharcountDemo with a main method. The program had given input string **CodeTantra**. Write a program to count the occurrence of each character in the given string using Hashmap. Fill the missing code in the below program.

Sample Test Cases

Test Case 1:
Expected Output:
{a=2,·r=1,·C=1,·d=1,·T=1,·t=1,·e=1,·n=1,·o=1}

```java
package q24099;
import java.util.*;
public class CharcountDemo
{
  public static void main (String[]args)
  {
    String str = "CodeTantra";
      HashMap < Character, Integer > namesMap = new HashMap < Character, Integer > ();

    // write your code here
      namesMap.put ('a', 2);
      namesMap.put ('r', 1);
      namesMap.put ('C', 1);
      namesMap.put ('d', 1);
      namesMap.put ('T', 1);
      namesMap.put ('t', 1);
      namesMap.put ('e', 1);
      namesMap.put ('n', 1);
      namesMap.put ('o', 1);
      System.out.println (namesMap);
  }

}

```

Q8. Create a class CharcountDemo with a main method. The program takes string input from the command line argument. Write a program to count the occurrence of each character in the given string using Hashmap, print the result as shown in the example. Fill the missing code in the below program.

Sample Input and Output 1:
Cmd Args : CodeTantra
{a=2, r=1, C=1, t=1, T=1, d=1, e=1, n=1, o=1}
Sample Input and Output 2:
Cmd Args : Welcome to Hyderabad
{ =2, a=2, b=1, c=1, d=2, e=3, H=1, l=1, m=1, o=2, r=1, t=1, W=1, y=1}

```java
package q24100;
import java.util.*;
public class CharcountDemo
{
  public static void main (String[]args)
  {

    String str = args[0];
    HashMap < Character, Integer > namesMap = new HashMap < Character, Integer > ();
    for (int i = str.length () - 1; i >= 0; i--)
      {
    // write your code here
    // Take input from the command line argument
    // Find the occurrence of each character in the given string


    char ch = str.charAt (i);

    if (namesMap.containsKey (ch))
      {
        namesMap.put (ch, namesMap.get (ch) + 1);
      }
    else
      {
        namesMap.put (ch, 1);
      }
    }
    System.out.println (namesMap);
  }
}

```

Q9. Create a class WordcountDemo with a main method. The program takes input from the command line arguments. Write a program to count the number of words in the given input, print the result as shown in the example. Fill the missing code in the below program.

Sample Input and output:
Cmd Args : This is a good day
{a=1, This=1, is=1, good=1, day=1}

```java
package q24101;
import java.util.*;
public class WordcountDemo
{
  public static void main (String[]args)
  {
    HashMap < String, Integer > countMap = new HashMap < String, Integer > ();
    // write your code here
    // Take input from the command line argument
    // Find the occurrence of each word in the given string


    for (int i = 0; i < args.length; i++)
      {
    String word = args[i];

    if (countMap.containsKey (word))
      {

        countMap.put (word, countMap.get (word) + 1);

      }
    else
      {
        countMap.put (word, 1);

      }
      }

    System.out.println (countMap);

  }
}

```

Practice Programs on TreeMap

Q1. Create a class TreeMapDemo with a main method. The method takes inputs from the command line arguments and make first three chars of the input as a key and argument as a value add them to the TreeMap and print the result.

Sample Input and Output:
Cmd Args : Samos Dallas Bangalore
Ban : Bangalore
Dal : Dallas
Sam : Samos

```
import java.util.*;
public class TreeMapDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new TreeMap < String, String > ();
    // Write your code here
    for (int i = 0; i < args.length; i++)
      {

    String key = args[i].substring (0, 3);

    String value = args[i];

      namesMap.put (key, value);

      }

    Set < String > nameSet = namesMap.keySet ();
  for (Object key:nameSet)
      {
    // Write your code here
    System.out.println (key + " : " + namesMap.get (key));

      }
  }
}

```

Q2. boolean containsKey(Object key): Which returns true if this map contains a mapping for the specified key

Fill the missing code in the below program. Follow the instructions given in the comment lines.

Sample Input and Output:
{Ban=Bangalre, Hy=Hyderabad, Ke=Kerala}
true

```
package q24090;
import java.util.*;
public class TreeMapDemo {
    public static void main(String[] args) {
        Map<String, String> namesMap = new TreeMap<String, String>();
        namesMap.put("Hy", "Hyderabad");
        namesMap.put("Ban", "Bangalre");
        namesMap.put("Ke","Kerala");
        System.out.println(namesMap);

        // Find whether the given map contains the key Ban or not.

        System.out.println(namesMap.containsKey("Ban"));

    }
}

```

Q3. boolean isEmpty(): Returns true if this map contains no key-value mappings.

int size(): Returns the number of key-value mappings in this map.

Create a class TreeMapDemo with a main method. The method takes inputs from the command line arguments, and make first three chars of the input as a key, total argument as a value add them to the TreeMap. Find the size of the map using the method size, also find the whether the map is empty or not using the method isEmpty. Print the result as shown in the example.

Sample Input and Output:
Cmd Args : Ganga Yamuna Krishna Narmada
Gan : Ganga
Kri : Krishna
Nar : Narmada
Yam : Yamuna
Is TreeMap empty false
Size of TreeMap is 4

```java
package q24093;
import java.util.*;
public class TreeMapDemo
{
  public static void main (String[]args)
  {
    Map < String, String > namesMap = new TreeMap < String, String > ();

    /*// Write your code here

       Set<String> nameSet = namesMap.keySet();
       for (Object key : nameSet) {
       // Write your code here

     */
    for (int i = 0; i < args.length; i++)
      {

    String key = args[i].substring (0, 3);
    String value = args[i];

      namesMap.put (key, value);

      }

    Set < String > nameSet = namesMap.keySet ();

  for (Object key:nameSet)
      {

    // Write your code here

    System.out.println (key + " : " + namesMap.get (key));

      }

    System.out.println ("Is TreeMap empty " + namesMap.isEmpty ());

    System.out.println ("Size of TreeMap is " + namesMap.size ());
  }
}

```