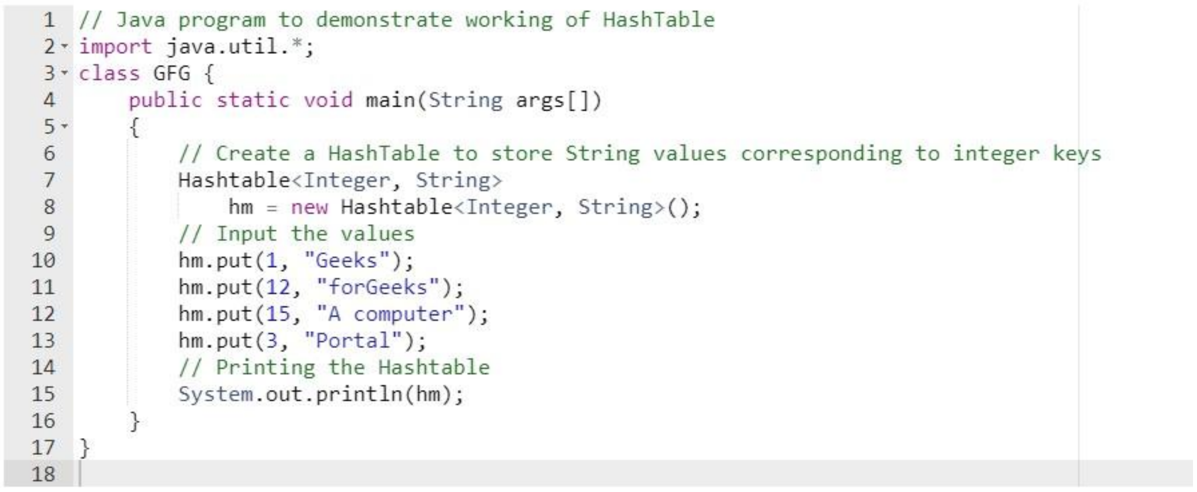
Implementing Hashing in Java

Java provides many built-in classes and interfaces to implement hashing easily. That is, without creating any HashTable or HashFunction. Java mainly provides us with the following classes to implement Hashing:

1. HashTable **(A synchronized implementation of hashing)**: 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.



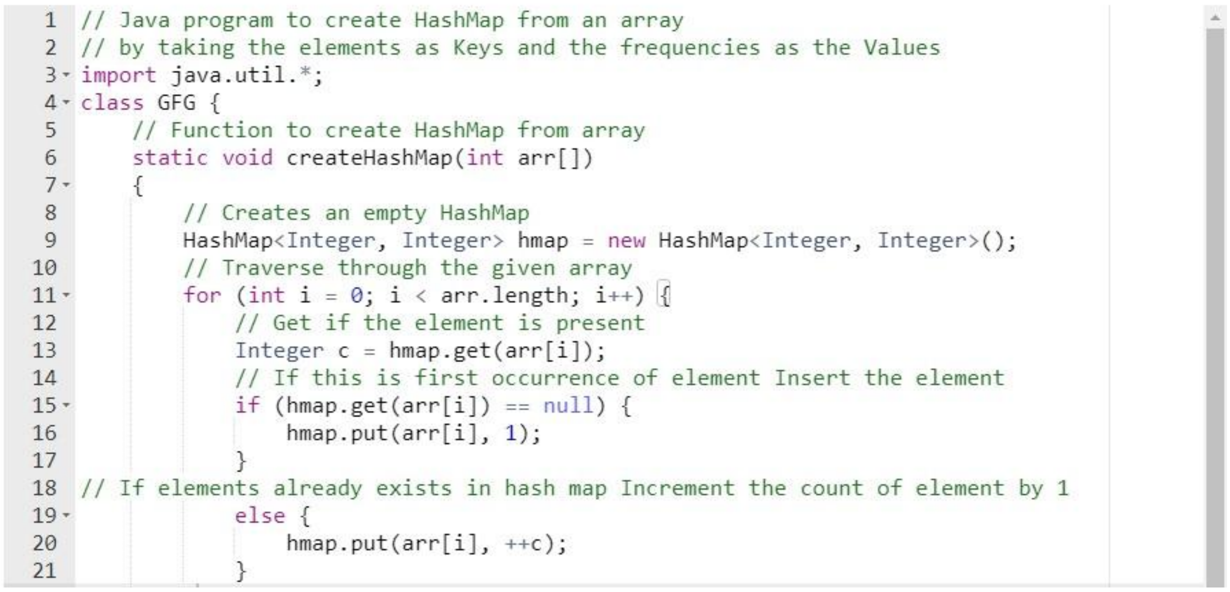
```java
// Java program to demonstrate working of HashTable
import java.util.*;
class GFG {
    public static void main(String args[])
    {
        // Create a HashTable to store String values corresponding to integer keys
        Hashtable<Integer, String>
            hm = new Hashtable<Integer, String>();
        // Input the values
        hm.put(1, "Geeks");
        hm.put(12, "forGeeks");
        hm.put(15, "A computer");
        hm.put(3, "Portal");
        // Printing the Hashtable
        System.out.println(hm);
    }
}

```

Output:

```
{15=A computer, 3=Portal, 12=forGeeks, 1=Geeks}
```

2. HashMap **(A non-synchronized faster implementation of hashing)**: HashMap are also similar to HashTables in Java but they are faster in comparison as they are not synchronised. HashMap are used to store key-value pairs or to map a given value to a given key. The general application of HashMaps is to count frequencies of elements present in an array or list.



```java
// Java program to create HashMap from an array
// by taking the elements as Keys and the frequencies as the Values
import java.util.*;
class GFG {
    // Function to create HashMap from array
    static void createHashMap(int arr[])
    {
        // Creates an empty HashMap
        HashMap<Integer, Integer> hmap = new HashMap<Integer, Integer>();
        // Traverse through the given array
        for (int i = 0; i < arr.length; i++) {
            // Get if the element is present
            Integer c = hmap.get(arr[i]);
            // If this is first occurrence of element Insert the element
            if (hmap.get(arr[i]) == null) {
                hmap.put(arr[i], 1);
            }
// If elements already exists in hash map Increment the count of element by 1
            else {
                hmap.put(arr[i], ++c);
            }
        }
        // Print HashMap
        System.out.println(hmap);
    }
    // Driver method to test above method
    public static void main(String[] args)
    {
        int arr[] = { 10, 34, 5, 10, 3, 5, 10 };
        createHashMap(arr);
    }
}
```

Output:

```
{34=1, 3=1, 5=2, 10=3}
```

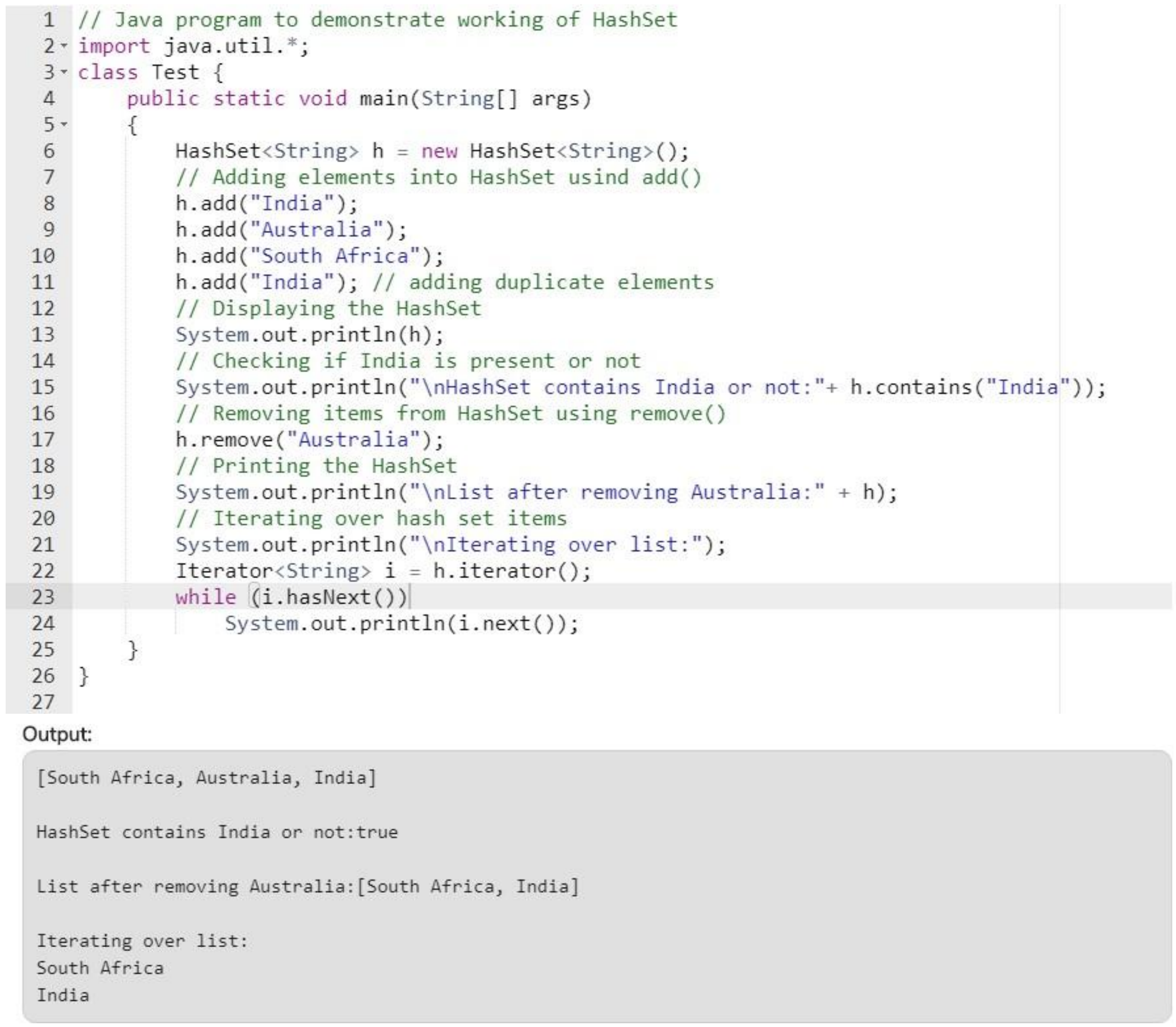
3. LinkedHashMap **(Similar to HashMap, but keeps order of elements)**:

```
// Java program to demonstrate working of LinkedHashMap
import java.util.*;
public class BasicLinkedHashMap
{
    public static void main(String a[])
    {
        LinkedHashMap<String, String> lhm = new LinkedHashMap<String, String>();
        lhm.put("one", "practice.geeksforgeeks.org");
        lhm.put("two", "code.geeksforgeeks.org");
        lhm.put("four", "quiz.geeksforgeeks.org");
        // It prints the elements in same order as they were inserted
        System.out.println(lhm);
        System.out.println("Getting value for key 'one': " + lhm.get("one"));
        System.out.println("Size of the map: " + lhm.size());
        System.out.println("Is map empty? " + lhm.isEmpty());
        System.out.println("Contains key 'two'? "+  lhm.containsKey("two"));
    System.out.println("Contains value 'practice.geeks"+"forgeeks.org'? "+
    lhm.containsValue("practice"+ ".geeksforgeeks.org"));
        System.out.println("delete element 'one': " +  lhm.remove("one"));
        System.out.println(lhm);
    }
}
```

Output:

```
{one=practice.geeksforgeeks.org, two=code.geeksforgeeks.org, four=quiz.geeksforgeeks.org}
Getting value for key 'one': practice.geeksforgeeks.org
Size of the map: 3
Is map empty? false
Contains key 'two'? true
Contains value 'practice.geeksforgeeks.org'? true
delete element 'one': practice.geeksforgeeks.org
{two=code.geeksforgeeks.org, four=quiz.geeksforgeeks.org}
```

4. HashSet **(Similar to HashMap, but maintains only keys, not pair)**: The HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance. The class also offers constant time performance for the basic operations like add, remove, contains and size assuming the hash function disperses the elements properly among the buckets. HashSets are generally used to keep a check on whether an element is present in a list or not.



```java
// Java program to demonstrate working of HashSet
import java.util.*;
class Test {
    public static void main(String[] args)
    {
        HashSet<String> h = new HashSet<String>();
        // Adding elements into HashSet usind add()
        h.add("India");
        h.add("Australia");
        h.add("South Africa");
        h.add("India"); // adding duplicate elements
        // Displaying the HashSet
        System.out.println(h);
        // Checking if India is present or not
        System.out.println("\nHashSet contains India or not:"+ h.contains("India"));
        // Removing items from HashSet using remove()
        h.remove("Australia");
        // Printing the HashSet
        System.out.println("\nList after removing Australia:" + h);
        // Iterating over hash set items
        System.out.println("\nIterating over list:");
        Iterator<String> i = h.iterator();
        while (i.hasNext())
            System.out.println(i.next());
    }
}
```

Output:

```
[South Africa, Australia, India]

HashSet contains India or not:true

List after removing Australia:[South Africa, India]

Iterating over list:
South Africa
India
```

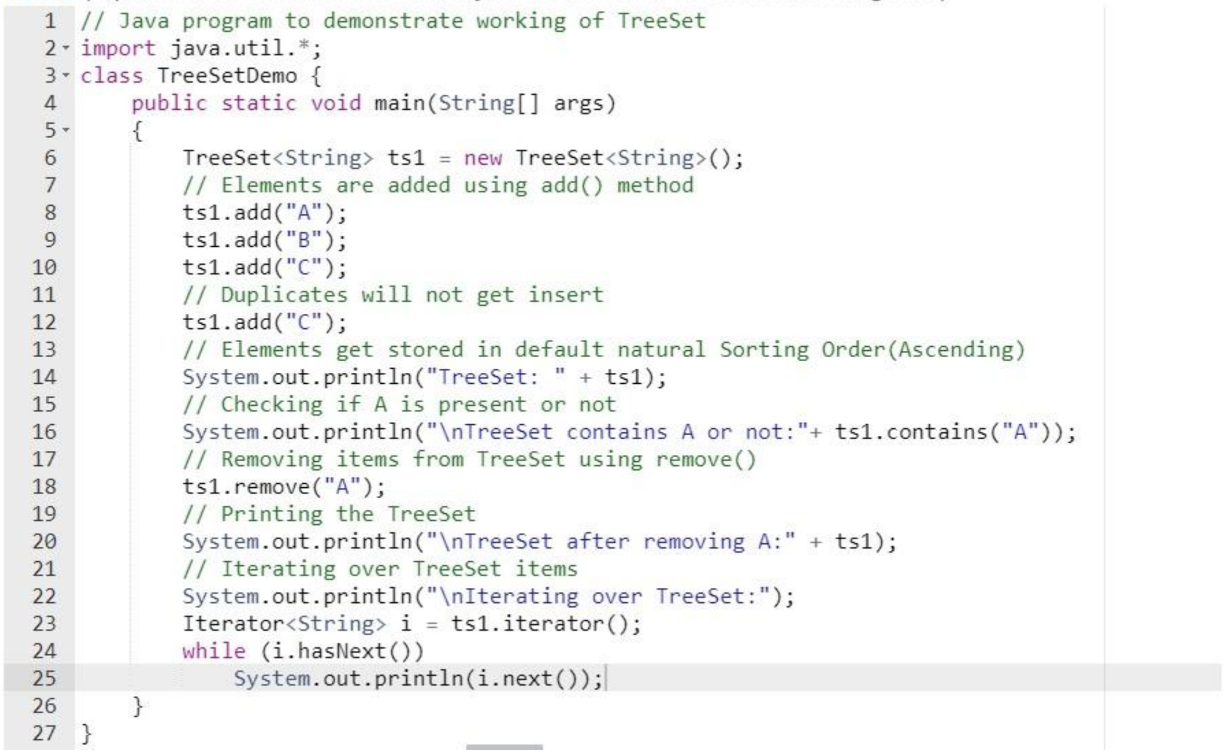
5. LinkedHashSet (Similar to LinkedHashMap, but maintains only keys, not pair):

```java
// Java program to demonstrate working of LinkedHashSet
import java.util.LinkedHashSet;
public class Demo
{
    public static void main(String[] args)
    {
        LinkedHashSet<String> linkedset = new LinkedHashSet<String>();
        // Adding element to LinkedHashSet
        linkedset.add("A");
        linkedset.add("B");
        linkedset.add("C");
        linkedset.add("D");
        // This will not add new element as A already exists
        linkedset.add("A");
        linkedset.add("E");
        System.out.println("Size of LinkedHashSet = " + linkedset.size());
        System.out.println("Original LinkedHashSet:" + linkedset);
        System.out.println("Removing D from LinkedHashSet: " +  linkedset.remove("D"));
        System.out.println("Trying to Remove Z which is not "+
                           "present: " + linkedset.remove("Z"));
        System.out.println("Checking if A is present=" +
                           linkedset.contains("A"));
        System.out.println("Updated LinkedHashSet: " + linkedset);
    }
}
```

Output:

```
Size of LinkedHashSet = 5
Original LinkedHashSet:[A, B, C, D, E]
Removing D from LinkedHashSet: true
Trying to Remove Z which is not present: false
Checking if A is present=true
Updated LinkedHashSet: [A, B, C, E]
```

6. TreeSet (Implements the SortedSet interface, Objects are stored in a sorted and ascending order):



```java
// Java program to demonstrate working of TreeSet
import java.util.*;
class TreeSetDemo {
    public static void main(String[] args)
    {
        TreeSet<String> ts1 = new TreeSet<String>();
        // Elements are added using add() method
        ts1.add("A");
        ts1.add("B");
        ts1.add("C");
        // Duplicates will not get insert
        ts1.add("C");
        // Elements get stored in default natural Sorting Order(Ascending)
        System.out.println("TreeSet: " + ts1);
        // Checking if A is present or not
        System.out.println("\nTreeSet contains A or not:"+ ts1.contains("A"));
        // Removing items from TreeSet using remove()
        ts1.remove("A");
        // Printing the TreeSet
        System.out.println("\nTreeSet after removing A:" + ts1);
        // Iterating over TreeSet items
        System.out.println("\nIterating over TreeSet:");
        Iterator<String> i = ts1.iterator();
        while (i.hasNext())
            System.out.println(i.next());
    }
}
```

Output:

```
TreeSet: [A, B, C]

TreeSet contains A or not:true

TreeSet after removing A:[B, C]

Iterating over TreeSet:
B
C
```