Hashtable, HashMap & TreeMap

The difference between Hashtable and HashMap is that methods of Hashtable are synchronized and the method of HashMap are not synchronized. As a result, processing data in HashMap is faster; but HashMap is not thread-safe.

While HashMap is a good choice for single threaded applications, Hashtable is a good choice for multithreaded applications.

NOTE: Both Hashtable and HashMap do not maintain the entries based on ***key*** in the order in which they were put into the collection. If the order of the ***key*** is to maintained then use TreeMap.

java.util.Hashtable

```
public class HashtableDemo {
      public static void main(String[] args) {
            Hashtable<String,String> currencyTable
                  = new Hashtable<>();

         currencyTable.put("INR", "Indian Rupee");
         currencyTable.put("LKR", "Sri Lanka Rupee");
         currencyTable.put("IQD", "Iraqi Dinar");
          currencyTable.put("BHD", "Bharain Dinar");
          currencyTable.put("USD", "US Dollar");
          currencyTable.put("SGD", "Singapore Dollar");

          System.out.println(currencyTable);

          String currDesc = currencyTable.get("SGD");
          System.out.println("Value for SGD is: " + currDesc);

          currDesc = currencyTable.get("XYZ");
          System.out.println("Value for XYZ is: " + currDesc);
          System.out.println("~".repeat(20));

          //Get all the keys as an java.util.Enumeration
          // Enumeration<K> keys() <- Method is present in Hashtable class and  is not part
          //    of Map interface
          Enumeration<String> enumKeys = currencyTable.keys();

          //Display all keys and their corresponding values
          String key = "";
```

```java
        String value = "";
        while(enumKeys.hasMoreElements()) {
           key = enumKeys.nextElement();
           value = currencyTable.get(key);
           System.out.println(key + " : " + value);
        }
        System.out.println("~".repeat(20));

        Set<String> keySet = currencyTable.keySet();
        System.out.println("keySet: "  + keySet);
        Iterator<String> keyItr = keySet.iterator();

        while(keyItr.hasNext()) {
           key = keyItr.next();
           value = currencyTable.get(key);
           System.out.println(key + " : " + value);
        }
        System.out.println("~".repeat(20));

        //Checking for existence of a key
        System.out.println("LKR key exists: "
                + currencyTable.containsKey("LKR"));
        System.out.println("AUD key exists: "
                + currencyTable.containsKey("AUD"));

      //Checking for existence of a key
        System.out.println("Indian Rupee exists: "
                + currencyTable.containsValue("Indian Rupee"));
        System.out.println("INDIAN RUPEE exists: "
                + currencyTable.containsValue("INDIAN RUPEE"));
        System.out.println("~".repeat(20));

        // Processing Hashtable data with Map.Entry
        // NOTE: Entry is an inner interface in Map interface
        Set<Map.Entry<String, String>> entrySet
                      = currencyTable.entrySet();
        Iterator<Entry<String, String>> entrySetItr
                      = entrySet.iterator();
        while(entrySetItr.hasNext()) {
           Entry<String, String> entry = entrySetItr.next();
           System.out.println(entry.getKey() +
                                        " : " + entry.getValue());
        }
      }
}
```

NOTE: From the above code except of keys() method which returns an instance of java.util.Enumeration all the other methods are available in Map, hence HashMap and TreeMap will support those methods.

Hashtable vs HashMap

There are several **differences between** HashMap **and** Hashtable in **Java**:

1. Hashtable is synchronized**,** whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.
2. Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
3. One of **HashMap's** subclasses is LinkedHashMap**,** so **if** you need predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

➔ Read the below web article to have a better understanding of how Hashtable is different from HashMap:

https://www.baeldung.com/hashmap-hashtable-differences

java.util.HashMap

If change the object instantiation from HashMap to TreeMap then the key-value pairs will be maintained in the order in which they were put into the collection.

```
public class HashMapDemo {
      public static void main(String[] args) {
        Map<String,String> currencyMap
                 = new HashMap<>();

         currencyMap.put("INR", "Indian Rupee");
         currencyMap.put("LKR", "Sri Lanka Rupee");
         currencyMap.put("IQD", "Iraqi Dinar");
        currencyMap.put("BHD", "Bharain Dinar");
        currencyMap.put("USD", "US Dollar");
        currencyMap.put("SGD", "Singapore Dollar");

        System.out.println(currencyMap);

        String currDesc = currencyMap.get("SGD");
        System.out.println("Value for SGD is: " + currDesc);

        currDesc = currencyMap.get("XYZ");
        System.out.println("Value for XYZ is: " + currDesc);
        System.out.println("~".repeat(20));
```

```java
        Set<String> keySet = currencyMap.keySet();
        System.out.println("keySet: "  + keySet);
        Iterator<String> keyItr = keySet.iterator();
        String key = "";
        String value = "";

        while(keyItr.hasNext()) {
           key = keyItr.next();
           value = currencyMap.get(key);
           System.out.println(key + " : " + value);
        }
        System.out.println("~".repeat(20));

        //Checking for existence of a key
        System.out.println("LKR key exists: "
                + currencyMap.containsKey("LKR"));
        System.out.println("AUD key exists: "
                + currencyMap.containsKey("AUD"));

      //Checking for existence of a key
        System.out.println("Indian Rupee exists: "
                + currencyMap.containsValue("Indian Rupee"));
        System.out.println("INDIAN RUPEE exists: "
                + currencyMap.containsValue("INDIAN RUPEE"));
        System.out.println("~".repeat(20));

        // Processing HashMap data with Map.Entry
        // NOTE: Entry is an inner interface in Map interface
        Set<Map.Entry<String, String>> entrySet
                      = currencyMap.entrySet();
        Iterator<Entry<String, String>> entrySetItr
                      = entrySet.iterator();
        while(entrySetItr.hasNext()) {
           Entry<String, String> entry = entrySetItr.next();
           System.out.println(entry.getKey() +
                                        " : " + entry.getValue());
        }
      }
}
```

NOTE: In a Map a key cannot be duplicated, but values can be duplicated.

Working with Set in Java Collections Framework:

```
public class SetDemo {
      public static void main(String[] args) {
            HashSet<Integer> hSet = new HashSet<>();
            LinkedHashSet<Integer> lHSet =
                        new LinkedHashSet<>();
            TreeSet <Integer> treeSet = new TreeSet<>();

            hSet.add(45);
            hSet.add(21);
            hSet.add(-4);
            hSet.add(0);
            hSet.add(78);
            hSet.add(99);

            lHSet.add(45);
            lHSet.add(21);
            lHSet.add(-4);
            lHSet.add(0);
            lHSet.add(78);
            lHSet.add(99);

            treeSet.add(45);
            treeSet.add(21);
            treeSet.add(-4);
            treeSet.add(0);
            treeSet.add(78);
            treeSet.add(99);

            System.out.println("HashSet doesn't maintain the order of inesrtion:");
            System.out.println("HashSet: " + hSet);
            System.out.println("~".repeat(20));

            System.out.println("LinkdeHashSet: " + lHSet);
            System.out.println("TreeSet: " + treeSet);
            System.out.println("~".repeat(20));

            SortedSet<Integer> sortSet = treeSet.tailSet(45);
            System.out.println("tailSet(E) always returns SorstedSet inclusive of E (45):  "
                                    + sortSet);
            System.out.println("~".repeat(20));

            NavigableSet<Integer> navSet = treeSet.tailSet(45, false);
            System.out.println("tailSet(E, false) returns NavigableSet exclusive of E (45): "
                                    + navSet);
```

```
        navSet = treeSet.tailSet(45, true);
        System.out.println("tailSet(E, true) returns NavigableSet inclusive of E (45): "
                                    + navSet);
    }
}
```

Concept of Set in Java Collections Framework:

1. A set does not allow duplicate values.
2. HashSet and LinkedHashSet accept one ***null*** value.
3. TreeSet does not accept even one ***null*** value.
4. HashSet does not maintain the order of element insertion, so retrieving based on order of insertion is not possible.
5. LinkedHashSet maintains the order of element insertion, so retrieving based on order of insertion is possible.
6. TreeSet sorts the elements as per their natural order (as implemented by compareTo() method of Comparable), or by the Comparator implementation provided at the time construction of TreeSet object.
7. Methods like add(), addAll(), contains(), containsAll(), remove(), removeAll(), size(), isEmpty() etc are common to all Set implementation to process data in them.
8. **Hashset internally uses Hashtable to store elements.**