FOUNDATIONS OF COMPUTER SCIENCE
PROGRAMMING TEST DATED 22.2.2017

Air Traffic

The map abstract data type, i.e. a collection of key/value mappings with unique key, is defined by the following interface, in which K and V are the parametric types:

```
public interface Map<K extends Comparable, V>
{
   /**
      Returns the value to which the specified key is mapped
      @param key the specified key
      @return the value to which the specified key is mapped, or null if this
              map contains no mapping for the key
   */
   V get(K key);

   /**
      Returns true if this map contains no key-value mappings
      @return true if this map is empty. otherwise false
   */
   boolean isEmpty();

   /**
      Returns an array containing the keys of the key/value mappings of this
      map. The array is sorted in ascending order according to the natural order
      of the keys
      @return a sorted array containing the keys of the key/value mappings of
             this map, if the map is not empty, otherwise an empty array
   */
   Comparable[] keySet();

   /**
      Associates the specified value with the specified key in this map. If this
      map previously contained a mapping for the key, the old value is replaced
      by the specified value
      @param key with which the specified value is to be associated
      @param value value to be associated with the specified key
      @return the previous value associated with key, or null if there was no
              mapping for key.
      @throws IllegalArgumentException if the specified key or value is null
   */
   V put(K key, V value);

   /**
      Removes the mapping for the specified key from this map if it is present
      @param key the specified key of the mapping to be removed
      @return the previous value associated with key, or null if there was no
              mapping for key.
   */
   V remove(K k);

   /**
      Returns the number of key-value mappings in this map
      @return the number of key-value mappings in this map
   */
   int size();

   /**
      Returns an array view of the values contained in this map
      @return an array containing the values of this map or an empty array if
              this map is empty
   */
   Object[] values();
}
```

Write the code of the class
`public class M <K extends Comparable, V> implements Map <K, V>`
whose constructor constructs an empty extended map.
During the code correction, the grade assigned to the class will consider the asymptotic time complexity of the put, get and remove methods.

Then complete the code of the extended map subclass below
public class ME <K extends Comparable, V extends Comparable> extends M <K, V>

```
{   /**
      checks if there is a mapping with the specified key in this extended map
      @param k the specified key
      @return true if there is a mapping with the specified key in this map,
              otherwise false
   */
   public boolean containsKey(K k) {...}

   /**
      checks if there is a mapping with the specified value in this extended map
      @param k the specified value
      @return true if there is a mapping with the specified value in this map,
              otherwise false
   */
   public boolean containsValue(V v) {...}


   /**
      Associates the specified value with the specified key in this map. If this
      map previously contained a mapping for the key, the old value is replaced
      by the specified value
      @param key with which the specified value is to be associated
      @param value value to be associated with the specified key
      @return the previous value associated with key, or null if there was no
              mapping for key.
      @throws IllegalArgumentException if the specified key or value is null
   */
   public V put (K k, V v)  { return super.put(k, v); }

   /**
      Returns a sorted array containing the values of this map without
      duplicated values
      @return sorted array containing the values of this map without
              duplicated values. The array is sorted in ascending order
              according to the natural order of the values.
   */
   public Comparable[] valueSet() {...}
}
```

During the correction the code will be tested with the commands below:

```
$ rm * .class
$ javac MEMain.java
$ java MEMain airlines.txt
```

where airlines.txt is the attached file and the MEMain.java class is a test class performing the following actions:

- creates an extended map
- reads the mappings from standard input and inserts them into the extended map
- prints the number of mappings in the extended map on standard output
- prints the mappings on standard output
- prints the mapping values on standard output
- prints the sorted mapping values without duplicates on standard output
- removes all mappings from the map printing them on standard output
- prints the number of mappings on standard output.

```
public class MEMain // Test class of the extended map
{
```

```
   public static void main(String[] args) throws java.io.IOException
   {
      ME<String,String> m = new ME<String, String>();
      java.util.Scanner in = new java.util.Scanner(new
                                 java.io.FileReader(args[0]));
      while (in.hasNextLine())
      {
         java.util.Scanner tok = new java.util.Scanner(in.nextLine());
         m.put(tok.next(), tok.next());
      }
      in.close();

      System.out.println("SIZE = " + m.size());

      System.out.print("ENTRIES: ");
      Object[] keys = m.keySet();
      for (Object e: keys)
         System.out.print(e + "/" + m.get((String)e) + " ");
      System.out.println();

      System.out.print("VALUES: ");
      Object[] values = m.values();
      for (Object e: values)
         System.out.print(e + " ");
      System.out.println();

      System.out.print("VALUESET: ");
      values = m.valueSet();
      for (Object e: values)
         System.out.print(e + " ");
      System.out.println();

      System.out.print("REMOVE: ");
      int i = 0;
      while (!m.isEmpty())
         System.out.print(keys[i] + "/" + m.remove((String)keys[i++]) + " ");
      System.out.println();

      System.out.println("SIZE = " + m.size());
   }
}
```

The attached OK.txt file reports the printout on standard output when the class MEMain runs and reads data from the airlines.txt file.

Generally, the code that gives rise to errors at compile-time is evaluated insufficient.

During the correction, the following maximum scores will be assigned to the methods:
M: get 5, put 5, remove 5, keySet 4, values 4 (Total class M = 23)
ME: containsKey 2, containsValue 2, valueSet 2 (Tot = 6)
one score can be awarded for the style and two score for the time complexity.
In the implementation of classes M and ME it is not permitted:

- to add public variables and / or public methods, (same for protected, without specifier access);
- to use classes from the Java Platform API except those of the package java.lang.

At the end of the test, leave the files produced and the files provided by the teacher in the working directory.
The files you code and deliver must contain as the first line a comment with the candidate's name and surname, roll number, date and PC station number.

FOUNDATIONS OF COMPUTER SCIENCE

Name _____Roll no. _____PC station ADT_____

[] I deliver the code consisting of the following files: [] M.java [] ME.java

[] I do not submit the paper and I withdraw from the exam.

Signature _____ _____