Foundations of Computer Science
Exam dated 28 January 2025
Programming Test

PARAMETRIC SORTED MULTIMAP WITH PUBLIC NON-PARAMETRIC MAPPINGS

The multimap abstract data type, a container of key-value associations where dublicate keys are allowed, is defined with the following interface

```
public interface SortedMultimap<K extends Comparable<K>,V> {
   /**Returns a value to which the specified key is mapped.
      @param key the key whose associated value is to be returned.
      @return a value to which the specified key is mapped, or null if this sorted
              multimap contains no mapping for the key.
   */ V find(K key);

   /**Returns true if this sorted multimap contains no key-value mappings.
      @return true if this sorted multimap contains no key-value mappings.
   */ boolean isEmpty();

   /**Associates the specified value with the specified key in this sorted multimap.
      @param key the key with which the specified value is to be associated.
      @param value the value to be associated with the specified key.
      @throw java.lang.IllegalArgumentException if the specified key or value is null.
   */ void insert(K key, V value);

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

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

   /**Returns a sorted array view of the keys contained in this sorted multimap.
      @return an array view of the sorted keys contained in this sorted multimap where
              keys are sorted in ascending order according to their natural order.
   */ Object[] sortedKeys();
}
```

where K and V are the parametric data types for the key and value.

The class to be used for the mappings is the following public class

```
public class Entry {
   // instance variables
   private Object key, value;

   /**Constructs a new mapping with the specified key and value.
      @param k the specified key of this mapping.
      @param v the specified value of this mapping.
   */ public Entry(Object k, Object v) { setKey(k); setValue(v); }

   /**Returns the key of this mapping.
      @return the key of this mapping.
   */ public Object getKey() { return key; }

   /**Returns the value of this mapping.
      @return the value of this mapping.
   */ public Object getValue() { return value; }

   /**Sets the key of this mapping with the specified key.
      @param k the specified key.
   */ public void setKey(Object k) { key = k; }
```

```
   /**Sets the value of this mapping with the specified value.
      @param v the specified value.
   */ public void setValue(Object v) { value = v; }
}
```

Code first the class **public class SM<K extends Comparable<K>,V> implements SortedMultimap<K,V>** that implements a sorted multimap where the constructor constructs an empty sorted multimap. Then code the following extended sorted multimap **public class ESM<K extends Comparable<K>,V extends Comparable<V>> extends SM<K,V>** where

- public ESM() constructs an empty extended sorted multimap
- public Object[] findAll(K key) returns an array containing all values associated with the specified key
- public int size() overrides the superclass method with a recursive implementation
- public Object[] sortedKeySet() returns an array with the keys of this extended sorted multimap sorted in ascending order according to their natural order and without duplicates.

In the code evaluation, the following commands will be used

```
$ javac SMTester.java
$ java SMTester cities.txt
$ javac ESMTester.java
$ java ESMTester cities.txt
```

Where the SMTester and ESMTester test classes are below and the cities.txt text file is attached.

```
public class SMTester {
   public static void main(String[] args) throws java.io.IOException {
      // check command line
      if (args.length < 1) {
         System.out.println("Use: java TestM filename");
         return;
      }

      // define new test sorted multimap
      SortedMultimap<String,String> m = new SM();

      // read mappings from input file and add them to sorted multimap
      java.util.Scanner in = new java.util.Scanner(new java.io.FileReader(args[0]));
      // read mappings
      while (in.hasNextLine()) {
         // read new line
         String s = in.nextLine();

         // decompose new line
         java.util.Scanner tok = new java.util.Scanner(s).useDelimiter("[#]+");
         String key = tok.next();
         String value = tok.next();
         tok.close();

         // add new mapping to test sorted multimap
         m.insert(key, value);                          // test of put method
      }
      in.close();

      // print size
      System.out.println("SIZE " + m.size());          // test of size method

      // print keys
      Object[] keys = m.sortedKeys();                   // test of sortedKeys method
      System.out.print("SORTED KEYS: ");
```

```
      for (Object k : keys) System.out. print(k + " ");
      System.out.println();

      // remove mappings
      System.out.print("MAPPINGS: ");
      int i = 0;                      // key array index
      while (!m.isEmpty()) {                          // test of isEmpty method
         // get key
         String key = (String)keys[i];

         // remove mapping
         while (m.find(key) != null) {                // test of find method
            String value = m.remove(key);             // test of remove method

            // print mapping
            System.out.print(key + "/" + value + " ");
         }

         // increment key array index
         i++;
      }
      System.out.println();

      // print size
      System.out.println("SIZE " + m.size());           // test of size method
   }
}
public class ESMTester {
   public static void main(String[] args) throws java.io.IOException {
      // check command line
      if (args.length < 1) {
         System.out.println("Use: java MainME filename");
         return;
      }

      // define new extended multimap
      ESM<String,String> m = new ESM();                     // test of constructor

      // read mappings from input file and add them to multimap
      java.util.Scanner in = new java.util.Scanner(new java.io.FileReader(args[0]));
      // read mappings
      while (in.hasNextLine()) {
         // read new line
         String s = in.nextLine();

         // decompose new line
         java.util.Scanner tok = new java.util.Scanner(s).useDelimiter("[#]+");
         String key = tok.next();
         String value = tok.next();
         tok.close();

         // add new mapping to multimap
         m.insert(key, value);                           // test of insert method
      }
      in.close();

      // print size
      System.out.println("SIZE " + m.size());            // test of size method

      // print duplicated key
```

```
      Object[] keys =  m.sortedKeys();                // test of sortedKeySet method
      System.out.print("KEYS: ");
      for (Object o : keys) System.out.print(o + " ");
      System.out.println();

      // print non duplicated key
      Object[] keySet =  m.sortedKeySet();             // test of sortedKeySet method
      System.out.print("KEYSET: ");
      for (Object o : keySet) System.out.print(o + " ");
      System.out.println();

      // test of method findAll
      System.out.println("FINDALL: ");
      for (int i = 0; i < keySet.length; i++) {
         Object[] values = m.findAll((String)keySet[i]);   // test of findAll method

         System.out.print("KEY " + keySet[i] + ": ");
         for (int j = 0; j < values.length; j++) System.out.print(values[j] + " ");
         System.out.println();
      }

      System.out.println("SIZE BEFORE REMOVING: " + m.size());

      // remove all mappings
      int i = 0;
      while (!m.isEmpty()) {
         String curKey = (String)keySet[i++];
         while (m.find(curKey) != null) m.remove(curKey);
      }

      // print size
      System.out.println("SIZE AFTER REMOVING: " + m.size());   // test of size method
   }
 }
```

The OKSM.txt and OKESM.txt text files display printouts on standard output when the Mtester and METester classes are run and read data from the cities.txt text file.
The following scores will be assigned in the code evaluation:
 SM: find 6, insert 6, remove 6, sortedKeys 6 (TOT 24),
 ESM: findAll 2, size 2, sortedKeySet 1 (TOT 5)
one point can be assigned for the style and two points for the time complexity of the find method when its asymptotic time complexity is better than O(n).
Code that reports compile-time errors is usually considered insufficient.
In the submitted code, the student may not
- use instance variables with access specifiers other than private
- add methods other than private
- use classes from the Java Platform API of packages other than java.lang.

Student files must contain a Java comment as the first line, including the student's first and last name, ID, date, and workstation number.

FOUNDATIONS OF COMPUTER SCIENCE

Name_____ID._____

Date _____Taliercio work station No._____
Undersign one of the following statements:

□ I submit the following files for evaluation:
( )SM.java ( ) ESM.java ( ) other_____

□ I withdraw from the programming test.

Signature_____