**Implement a frequency counter.**

The frequency counter must insert data in sorted way.

It should handle int, char, float, double, String.

I have written program with comments for this problem. So reading the comments will help.

**package** com.stat;

**import** java.util.Iterator;

**import** java.util.Map.Entry;

**import** java.util.TreeMap;

/\*\*

\* Frequency class is used to generate frequency of character,

\* double, int, float, String.

\* \*/

**public** **class** Frequency {

/\*\*

\* We use TreeMap<String, Long> to store the key as String

\* and value i.e frequency in Long.

\* \*/

TreeMap<String, Long> freq;

**private** **static** **final** **long** ***INCREMENT\_VALUE*** = 1;

**private** **static** **final** **long** ***ZERO\_VALUE*** = 0;

/\*\*

\* Constructor used to built new TreeMap<>

\* \*/

**public** Frequency() {

freq = **new** TreeMap<String, Long>();

}

/\*\*

\* clear() method is used to clear all mapping in TreeMap<>

\* After this method returns the Map is Empty.

\* \*/

**public** **void** clear() {

freq.clear();

}

/\*\*

\* Used to add char data into the TreeMap<>

\* \*/

**public** **void** add(**char** value) {

addInTable(convert(value));

}

/\*\*

\* Convert char data to string using String.valueOf(char).

\* String.valueOf(char) returns new String.

\* \*/

**private** String convert(**char** value) {

**return** String.*valueOf*(value);

}

/\*\*

\* Used to add double data into the TreeMap<>

\* \*/

**public** **void** add(**double** value) {

addInTable(convert(value));

}

/\*\*

\* Convert double data to string using String.valueOf(double).

\* String.valueOf(double) returns new String.

\* \*/

**private** String convert(**double** value) {

**return** String.*valueOf*(value);

}

/\*\*

\* Used to add int data into the TreeMap<>

\* \*/

**public** **void** add(**int** value) {

addInTable(convert(value));

}

/\*\*

\* The purpose of converting int to double is that 1.00 is equal to 1.

\* So we convert float and int to double to maintain consistency.

\* Convert double data to string using String.valueOf(double).

\* String.valueOf(double) returns new String.

\* \*/

**private** String convert(**int** value) {

**return** String.*valueOf*(**new** Double(value));

}

/\*\*

\* Float will be converted to double to maintain the consistency in table.

\* Inturn that double will be converted to String.

\* \*/

**public** **void** add(**float** value) {

addInTable(convert(value));

}

/\*\*

\* Conversion of float to double and then to String.

\* \*/

**private** String convert(**float** value) {

**return** String.*valueOf*(**new** Double(value));

}

**public** **void** add(String value) {

addInTable(value);

}

/\*\*

\* This method is used to preserve the

\* K is String

\* V is count for K

\* \*/

**public** **void** addInTable(String value) {

/\*\*

\* K is String

\* V is count of K

\* \*/

**if** (freq.containsKey(value)) {

**long** previousCount = freq.get(value);

freq.put(value, previousCount + ***INCREMENT\_VALUE***);

} **else** {

freq.put(value, ***INCREMENT\_VALUE***);

}

}

/\*\*

\* Returns the count of the value from TreeMap<>

\* Convert to string and use getCount(String).

\* \*/

**public** **long** getCount(**char** value) {

**return** getCount(convert(value));

}

/\*\*

\* Returns the count of the value from TreeMap<>

\* Convert to string and use getCount(String).

\* \*/

**public** **long** getCount(**int** value) {

**return** getCount(convert(value));

}

/\*\*

\* Returns the count of the value from TreeMap<>

\* Convert to string and use getCount(String).

\* \*/

**public** **long** getCount(**float** value) {

**return** getCount(convert(value));

}

/\*\*

\* Returns the count of the value from TreeMap<>

\* Convert to string and use getCount(String).

\* \*/

**public** **long** getCount(**double** value) {

**return** getCount(convert(value));

}

/\*\*

\* Returns the count of the value from TreeMap<>

\* return the long type.

\* \*/

**public** **long** getCount(String value) {

**if** (!isEmpty()) {

**if** (freq.containsKey(value)) {

**return** freq.get(value).longValue();

} **else** {

**return** ***ZERO\_VALUE***;

}

} **else** {

**return** ***ZERO\_VALUE***;

}

}

/\*\*

\* Get total keys from TreeMap<>.

\* \*/

**public** **int** getKeysCount() {

**return** freq.keySet().size();

}

/\*\*

\* This method used to return total number of values from the TreeMap<>.

\* \*/

**public** **long** getValuesCount() {

Iterator<Long> iter = freq.values().iterator();

Long values = 0L;

**while** (iter.hasNext()) {

values = values + iter.next();

}

**return** values.longValue();

}

/\*\*

\* gets total key percent for the value

\* convert the value to string and call getKeyPercent()

\* \*/

**public** **double** getKeyPercent(**char** value) {

**return** getKeyPercent(convert(value));

}

/\*\*

\* gets total key percent for the value

\* convert the value to string and call getKeyPercent()

\* \*/

**public** **double** getKeyPercent(**int** value) {

**return** getKeyPercent(convert(value));

}

/\*\*

\* gets total key percent for the value

\* convert the value to string and call getKeyPercent()

\* \*/

**public** **double** getKeyPercent(**float** value) {

**return** getKeyPercent(convert(value));

}

/\*\*

\* gets total key percent for the value

\* convert the value to string and call getKeyPercent()

\* \*/

**public** **double** getKeyPercent(**double** value) {

**return** getKeyPercent(convert(value));

}

/\*\*

\* gets total key percent for the value

\* convert the value to string and call getKeyPercent()

\* return double value.

\*

\* To eliminate the threat of ArithmeticException(divide by zero error)

\* we check if TreeMap<> is empty of not.

\*

\* If the TreeMap<> is empty then return 0.

\* else compute the percentage.

\* \*/

**public** **double** getKeyPercent(String value) {

**if** (!isEmpty()) {

**if** (freq.containsKey(value)) {

**return** ((**double**) freq.get(value) / getValuesCount());

} **else** {

**return** ***ZERO\_VALUE***;

}

} **else** {

**return** ***ZERO\_VALUE***;

}

}

/\*\*

\* return the iterator() of entrySet of TreeMap<>

\* \*/

**public** Iterator<Entry<String, Long>> frequencyIterator() {

**return** freq.entrySet().iterator();

}

/\*\*

\* This method is useful for inserting another Frequency

\* class's object into current object.

\* \*/

**public** **void** insertFrequencies(Frequency frequency) {

//Get iterator for frequency

Iterator<Entry<String, Long>> iter = frequency.frequencyIterator();

**while** (iter.hasNext()) {

Entry<String, Long> entry = iter.next();

String K = entry.getKey();

Long V = entry.getValue();

**if** (freq.containsKey(K)) {

freq.put(K, freq.get(K) + V);

} **else** {

freq.put(K, V);

}

}

//System.out.println(freq);

}

/\*\*

\* return

\* true if TreeMap<> is empty

\* false if TreeMap<> is not empty.

\* \*/

**public** **boolean** isEmpty() {

**return** freq.isEmpty();

}

**public** String toString() {

**return** freq.toString();

}

**public** **static** **void** main(String[] args) {

Frequency f = **new** Frequency();

f.add(1);

f.add('a');

f.add('a');

f.add("aaa");

f.add("aaa");

f.add("aaa");

f.add(1.0f);

f.add(1.0);

System.***out***.println(f);

Frequency f1 = **new** Frequency();

f1.add(1);

f1.add('a');

f1.add("aaa");

f1.add("aaa");

f1.add("aaa");

f1.add(1.0f);

f1.add(1.0);

f1.insertFrequencies(f);

System.***out***.println(f1);

}

}

Ouput

{1.0=3, a=2, aaa=3}

{1.0=6, a=3, aaa=6}