****

Algorithmique

Informatique, Classe I1A

**S06 : Type Abstrait ‘’Ensemble’’ et ‘’Dictionnaire’’**

Date : 22/03/2018

Noms : Luke Perrottet, Stefan Pahud

Table des matières

[1 Ex. 1 2](#_Toc509496648)

[2 Ex. 2 2](#_Toc509496649)

[2.1 Code 2](#_Toc509496650)

[2.1.1 ShortToStringMap.java 2](#_Toc509496651)

[2.1.2 ShortToStringMapItr.java 4](#_Toc509496652)

[2.2 Résultat 5](#_Toc509496653)

[3 Ex. 3 5](#_Toc509496654)

[3.1 Code 5](#_Toc509496655)

[3.2 Résultat 6](#_Toc509496656)

[3.3 Question 6](#_Toc509496657)

# Ex. 1

The method ***Z*** returns a SetOfShorts that contains all the values of the recieved ***S*** that are smaller than ***X***

# Ex. 2

## Code

### ShortToStringMap.java

#### Version avec des objects

**package** s06;

**public** **class** **ShortToStringMap** {

**private** **int** size;

**protected** MapElt[] map;

**private** **static** **final** **int** DEFAULT\_SIZE = **10**;

**class** **MapElt** {

**private** **short** key;

**private** String val;

**public** **MapElt**() {

}

**public** **MapElt**(**short** key, String val) {

**this**.key = key;

**this**.val = val;

}

**public** **short** **getKey**() {

**return** key;

}

**public** String **getVal**() {

**return** val;

}

**public** **void** **setVal**(String val) {

**this**.val = val;

}

}

// ------------------------------

// Private methods

// ------------------------------

**private** **void** **checkMapFull**() {

**if** (size == map.length) {

MapElt[] newMap = **new** MapElt[map.length \* **2**];

**for** (**int** i = **0**; i < map.length; i++) {

newMap[i] = map[i];

}

map = newMap;

}

}

// return -1 if key doesn't exist

**private** **int** **getKeyIndex**(**short** key) {

**for** (**int** i = **0**; i < size; i++) {

**if** (map[i].getKey() == key)

**return** i;

}

**return** -**1**;

}

// ------------------------------

// Public methods

// ------------------------------

// ------------------------------------------------------------

**public** **ShortToStringMap**() {

map = **new** MapElt[DEFAULT\_SIZE];

}

// ------------------------------------------------------------

// adds an entry in the map, or updates it

**public** **void** **put**(**short** key, String img) {

**int** keyIndex = getKeyIndex(key);

**if** (keyIndex == -**1**) { // add

checkMapFull();

map[size++] = **new** MapElt(key, img);

} **else** { // update

map[keyIndex].setVal(img);

}

}

// ------------------------------------------------------------

// returns null if !containsKey(key)

**public** String **get**(**short** key) {

**int** keyIndex = getKeyIndex(key);

**if** (keyIndex != -**1**)

**return** map[keyIndex].getVal();

**return** **null**;

}

// ------------------------------------------------------------

**public** **void** **remove**(**short** e) {

**int** keyIndex = getKeyIndex(e);

**if** (keyIndex != -**1**) {

map[keyIndex] = map[--size];

}

}

// ------------------------------------------------------------

**public** **boolean** **containsKey**(**short** k) {

**return** **getKeyIndex**(k) != -**1**;

}

// ------------------------------------------------------------

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

**return** **size**() == **0**;

}

**public** **int** **size**() {

**return** size;

}

// ------------------------------------------------------------

// a.union(b) : a becomes "a union b"

// values are those in b whenever possible

**public** **void** **union**(ShortToStringMap m) {

**for** (**int** i = **0**; i < m.size; i++) {

put(m.map[i].getKey(), m.map[i].getVal());

}

}

// ------------------------------------------------------------

// a.intersection(b) : "a becomes a intersection b"

// values are those in b

**public** **void** **intersection**(ShortToStringMap s) {

**for** (**int** i = size - **1**; i >= **0**; i--) {

**short** key = map[i].getKey();

**if** (!s.containsKey(key))

remove(key);

**else**

**put**(key, s.get(key));

}

}

// ------------------------------------------------------------

// a.toString() returns all elements in

// a string like: {3:"abc",9:"xy",-5:"jk"}

**public** String **toString**() {

String s = "{";

**for** (**int** i = **0**; i < size; i++) {

s += map[i].getKey() + ":\"" + map[i].getVal() + "\",";

}

**if** (s.length() > **1**)

s = s.substring(**0**, s.length() - **1**); // remove last comma

s += "}";

**return** s;

}

}

#### Version avec deux tableaux

### ShortToStringMapItr.java

**package** s06;

**public** **class** **ShortToStringMapItr** {

**private** ShortToStringMap m;

**private** **int** currIndex;

// ----------------------------------------

**public** **ShortToStringMapItr**(ShortToStringMap m) {

**this**.m = m;

currIndex = **0**;

}

// ----------------------------------------

**public** **boolean** **hasMoreKeys**() {

**return** currIndex < (m.size() - **1**);

}

// ----------------------------------------

// PRE-condition: hasMoreKeys()

**public** **short** **nextKey**() {

**return** m.map[currIndex++].getKey();

}

}

## Résultat

Using seed **970**

Add/remove seems OK

Iterator seems OK

Union seems OK

Intersection seems OK

Test passed successfully !

# Ex. 3

## Code

**package** s06;

**import** **java.util.Random**;

**import** **java.util.Arrays**;

**public** **class** **RandomTable** {

**static** Random r = **new** Random();

// ------------------------------------------------------------

**public** **static** **short**[] **randomTable**(**short** m, **short** n) {

**short**[] result = **null**;

// PSEUDO-CODE :

// créer un ensemble vide

// tant que l'ensemble a moins de m éléments

// ajouter à l'ensemble un élément au hasard

// à l'aide d'un itérateur, copier l'ensemble dans un tableau

// trier le tableau

SetOfShorts randT = **new** SetOfShorts();

SetOfShortsItr itr = **new** SetOfShortsItr(randT);

**while** (randT.size() < m) {

randT.add((**short**) r.nextInt(n));

}

result = **new** **short**[m];

**int** i = **0**;

**while** (itr.hasMoreElements()) {

result[i++] = itr.nextElement();

}

Arrays.sort(result);

**return** result;

}

// ------------------------------------------------------------

**static** **void** **testRandomTable**(**short** m, **short** n) {

**short**[] s = randomTable(m, n);

**int** i;

**if** (m != s.length)

**throw** **new** **RuntimeException**("Size of array is not correct");

**if** (s.length > **0** && s[**0**] < **0**)

**throw** **new** **RuntimeException**("Elements must be in [0..n[");

**for** (i = **0**; i < s.length - **1**; i++) {

**if** (s[i] >= s[i + **1**])

**throw** **new** **RuntimeException**(

"Array should be sorted and contain distinct numbers\n[" + stringFromArray(s) + "]");

}

System.out.println("\nTest passed successfully !");

**for** (i = **0**; i < m; i++)

System.out.print(" " + s[i]);

}

// ------------------------------------------------------------

**static** String **stringFromArray**(**short**[] s) {

String str = "";

**for** (**int** i = **0**; i < s.length; i++) {

str = str + s[i] + " ";

}

**return** str;

}

// ------------------------------------------------------------

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

**short** m = **10**;

**short** n = **50**;

**if** (args.length == **2**) {

m = Short.parseShort(args[**0**]);

n = Short.parseShort(args[**1**]);

}

testRandomTable(m, n);

}

}

## Résultat

Test passed successfully !

**0** **2** **3** **4** **14** **28** **35** **36** **42** **47**

## Question

Résoudre ce problème sans ensemble, par deux autres algorithmes :

* Avec une pile, où on push() des valeurs randoms jusqu’à m éléments, pour ensuite les pop() dans le tableau.
* Avec une liste et un itérateur, où on insertAfter() des valeurs randoms jusqu’à m éléments, pour ensuite faire une boucle depuis le premier élément jusqu’à isLast() pour insérer les valeurs dans le tableau avec consultAfter().