Nom : _____

Prénom : _____

Groupe : _____

Identifiant : _____

/ 50

Haute École de Bruxelles
École Supérieure d'Informatique
Bachelor en Informatique

2015 – 2016

**Développement – 1re**

Examen

# Statistiques et tri par énumération de la population

**Consignes**

Pour l'ensemble de l'examen :

- vous avez 4 heures de temps ;
- vous gérerez ce temps à votre meilleure convenance pour réaliser les solutions en algorithmique et en Java.

Le but de cet examen est de trier et d'extraire des statistiques d'un long tableau représentant les âges (sous la forme d'un entier) d'une population d'êtres humains. Comme ce tableau peut être très long (par exemple toute la population d'un pays), il faudra veiller à ce que les opérations (dont le tri) se fassent de manière efficace pour ne pas que votre programme soit trop lent.

L'énoncé ci-dessous vous guidera pas à pas dans l'élaboration de petits algorithmes simples, qui seront ensuite appelés dans un algorithme principal à la fin.



Version du document : 12 janvier 2016

# I
# Algorithmique

**Consignes**

Pour la partie algorithmique :

- vous ne pouvez pas utiliser de notes ;
- vos réponses se feront au bic bleu ou noir sur la feuille de réponses ;
- sauf spécification contraire, les données lues ou reçues ne comportent pas d'erreurs ;
- les noms en gras (variables et types) doivent être respectés ;
- veillez à travailler de manière modulaire.

## 1 Génération aléatoire du tableau (3 points)

Écrivez un algorithme **générerTableau** qui reçoit en paramètre un âge maximal `âgeMax` (entier) et un entier `n`, et qui retourne un tableau de taille `n` dont chaque case contient un entier entre 0 et `âgeMax`. Pour rappel, vous disposez de la fonction `hasard(m: entier)` qui retourne un nombre entier entre 1 et `m`.

```
module genererTableau(ageMax↓ : entier, n : entier) → tableau de n entiers
    ages : tableau de n entiers
    pour i de 0 a n-1 faire
        ages[i] ← hasard(ageMax+1)-1
    fin pour
fin module
```

## 2 Compter la répartition de la population par âge (6 points)

Écrivez un algorithme **répartionÂges** qui reçoit un tableau `âges` de `n` entiers (non-trié) et l'âge maximum de la population.

L'algorithme retourne un tableau de comptage des âges. Ce nouveau tableau contiendra, à la case $i$, le nombre de personnes ayant l'âge $i$ (on tiendra compte de tous les âges, y compris l'âge 0 des nouveaux-nés). Ce tableau est de taille « âge maximum ».

Par exemple, si le tableau `âges` contient 500 personnes de 34 ans, alors le tableau de comptage de la répartition des âges contiendra la valeur 500 à l'indice 34 (i.e. `comptage[34]=500`).

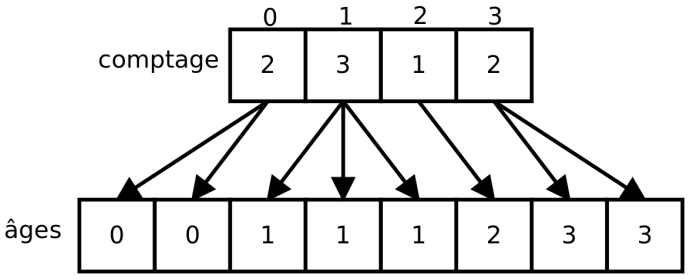
```
module repartionAges(ages↓ : tableau de n entiers, ageMax↓ : entier) → tableau de
ageMax+1 entiers
    comptage : tableau de ageMax+1 entiers
    age : entier
    comptage ← {0, . . . , 0}
    pour i de 0 a n-1 faire
        age ← ages[i]
        comptage[age] ← comptage[age]+1
    fin pour
    retourner comptage
fin module
```

## 3 Tri par énumération (9 points)

Nous voulons à présent trier notre tableau. Nous allons utiliser la technique suivante (qui est plus rapide que le tri par insertion et le tri par sélection) : on part du tableau de comptage de la répartition des âges de la question 2, et pour chaque indice `i` de ce tableau, on rajoute `comptage[i]` fois la valeur `i` dans le tableau à trier.

Soit le tableau à trier $\hat{a}ges$ : $2, 1, 0, 3, 3, 1, 0, 1$.

La figure ci-dessous illustre cette idée pour le tableau `âges` contenant des nombres entre 0 et 3.




Écrivez un algorithme **trier**, qui recevra en paramètre le tableau `âges` des âges à trier, ainsi que le tableau `comptage` du comptage de la répartition des âges calculé à la question 2. L'algorithme remplira le tableau `âges` avec les bonnes valeurs, dans l'ordre, en écrasant les anciennes valeurs.

```
module trier(ages↓↑ : tableau de n entiers, comptage↓ : tableau de m entiers)
    k, cpt : entier
    k ← 0
    pour i de 0 a m-1 faire
        cpt ← comptage[i]
        pour j de 1 a cpt faire
            ages[k] ← i
            k ← k+1
        fin pour
    fin pour
fin module
```

## 4 Statistiques (7 points)

Écrivez un algorithme **statistiques** qui va effectuer les opérations suivantes. Il devra bien entendu faire appel aux algorithmes écrits précédemment.

- Créer dans un premier temps un tableau `âges` de taille 50.000 (cinquante mille) et contenant des âges aléatoires entre 0 et 120 ans.
- Utiliser l'algorithme des questions précédentes afin de trier le tableau. Afficher son contenu une fois trié.
- Afficher le nombre de mineurs (individus de strictement moins de 18 ans), le nombre de centenaires (individus d'au moins 100 ans) et le nombre d'individus qui sont ni mineurs ni centenaires. Veillez à réutiliser le tableau de comptage de la question 2, afin que votre code soit le plus rapide possible.

— N'hésitez pas à découper la solution de cette question en petits algorithmes, pour rendre votre code plus clair et plus modulaire.

```
module statistiques()
    ages : tableau de 1000000 entiers,
    comptage : tableau de 121 entiers
    max, mineurs, majeurs : entier
    ages ← genererTableau(120, 1000000)
    comptage ← repartitionAges(ages, 120)
    trier(ages, comptage)
    afficherTab(ages)
    mineurs ← nbMineurs(comptage)
    centenaires ← nbCentenaies(comptage)
    afficher "Nombre de mineurs : ", mineurs, "Nombre de centenaires : ", centenaires,
"Autres : ", 1000000-mineurs-centenaires
fin module

module afficherTab(tab↓ : tableau de n entiers)
    pour i de 0 a n-1 faire
        afficher tab[i]
    fin pour
fin module

module nbMineurs(comptage↓ : tableau de n entiers) → entier
    somme : entier
    somme ← 0
    pour i de 0 a 17 faire
        somme ← somme+comptage[i]
    fin pour
    retourner somme
fin module

module nbCentenaires(comptage↓ : tableau de n entiers) → entier
    somme : entier
    somme ← 0
    pour i de 100 a n-1 faire
        somme ← somme+comptage[i]
    fin pour
    retourner somme
fin module
```

# II
# Java et laboratoire

**Consignes**

Pour la partie java,

- *Vous réaliserez votre travail sur* `linux1` *et le déposerez dans le casier linux de votre professeur par la commande* `casier`.
- *Vous disposez de toutes vos notes ainsi que de l'aide en ligne.*
- *Il ne suffit pas que votre code compile. Testez-le pour identifier d'éventuelles erreurs à l'exécution.*
- *La cotation tiendra compte aussi du style de programmation que vous avez acquis.*
- *Respectez bien les noms de package, classe, méthodes demandés dans l'énoncé.*
- *Vous remplacerez bien sûr g12345 par votre numéro d'étudiant.*

## 5 Question préalable (*pré-condition*)

Créez un répertoire `evaluations/janvier`. Changez les droits sur votre répertoire `janvier` pour donner les permissions de lecture et d'exécution aux professeurs mais aucun droit aux autres étudiants. Appelez votre professeur pour lui montrer que vos permissions ont bien été changées.

Vous ne continuerez pas l'examen tant que cette question n'a pas été validée par votre professeur.

## 6 Mise en place (3 points)

Dans la suite, votre classe s'appellera `g12345.evaluation.janvier.Population`[1].

Écrivez ici (l'important dans les questions qui suivent est la cohérence de l'ensemble) :

— votre répertoire de travail (probablement `~/evaluations/<votre choix>`) ;

— l'instruction que doit contenir votre classe pour faire partie du package demandé ;

1. Il devrait être évident qu'il faut remplacer 12345 par **votre** identifiant.

— la commande (complète et précise) que vous utilisez (à partir de votre répertoire courant) pour **compiler** votre classe ;

— la commande (complète et précise) que vous utilisez (à partir de votre répertoire courant) pour **exécuter** votre classe ;

— le contenu minimal de votre variable d'environnement `CLASSPATH`

## 7 Premières méthodes (6 points)

Écrivez les méthodes `générerTableau` et `répartitionÂges` comme décrites dans la partie algorithmique.

## 8 Tests JUnit ( 6 points)

Écrivez au minimum 5 tests JUnit permettant de tester la validité de la méthode `répartitionÂges`.

Ces tests se trouveront dans une classe `TestPopulation`.

Écrivez ici :

— la commande permettant de lancer les tests JUnit [2] ;

— le contenu de la variable d'environnement `CLASSPATH` pour pouvoir exécuter ces tests ;

— une commande permettant de **rediriger** les résultats des tests dans le fichier `tests.log`.

2. Inutile d'écrire ici un `alias`

## 9 Les méthodes suivantes (6 points)

### i La méthode de tri

Écrivez la méthode de tri par énumération simplifiée comme décrite dans la partie algorithmique.

— `public static void trier(int[] âges, int[] comptage);`

### ii Une méthode principale

Écrivez une méthode principale réalisant les tâches décrites dans la partie « statistiques ».

## 10 Javadoc (4 points)

Si vous ne l'avez pas fait au fur et à mesure, écrivez la **javadoc** pour vos méthodes publiques et générez-la dans un sous-répertoire `doc`.

```java
package pbt.evaluation.dev1.janvier;

import java.util.Random;
import java.util.Arrays;

public class Population {

    /**
     * Écrivez un algorithme générerTableau.
     * Cette méthode retourne un tableau de taille n dont chaque
     * case contient un entier entre 0 et âgeMax.
     *
     * @param âgeMaximum l'âge maximum pouvant apparaître dans le tableau
     * @param n le nombre d'âges dans le tableau
     * @return un tableau de n âges
     */
    public static int[] générerTableau(int âgeMaximum, int n){
        int[] âges = new int[n];
        final Random R = new Random();

        for(int i=0; i < âges.length; i++){
            âges[i] = R.nextInt(âgeMaximum);
        }
        return âges;
    }

    /**
     * Écrivez un algorithme répartitionÂges.
     *
     * L¿algorithme retourne un tableau de répartition des âges.
     * Ce nouveau tableau contiendra, à la case i, le nombre de personnes ayant
     * l¿âge i (on tiendra compte de tous les âges, y compris l¿âge 0 des
     * nouveaux-nés).
     *
     * Par exemple, si le tableau âges contient 500 personnes de 34 ans,
     * alors le tableau de comptage contiendra la valeur 500 à l¿indice 34
     * (i.e. comptage[34]=500).
     *
     * Remarque: Le méthode ne vélifie pas que tous les âges sont en-dessous
     * de l'âge maximum.
     *
     * @param âges le tableau des âges de la population
     * @param âgeMaximum l'âge maximum de la population
     * @return le tableau contenant la répartition des âges
     */
    public static int[] répartitionÂges(int[] âges, int âgeMaximum){
        if(âges == null) {
            throw new IllegalArgumentException("Tableau null");
        }

        int[] répartition = new int[âgeMaximum + 1];

        for(int i=0; i < âges.length; i++){
            répartition[âges[i]] += 1;
        }
        return répartition;
    }

    /**
     * Tri par énumération.
     *
     * Mise en ¿uvre du tri par énumération simplifié dans le cas de
     * nombre entiers.
     *
     * @param âges le tableau à trier
     * @param comptage le tableau contenant la répartition des âges
     * @return le tableau trié
     */
    public static int[] triÉnumération(int[] âges, int[] comptage){
        int iâges = 0;
        for(int i=0; i < comptage.length; i++){
            for(int j=0; j < comptage[i]; j++){
                âges[iâges++] = i;
```

```
            }
        }
        return âges;
    }

    // Ce main correspond à l'algo statistiques
    public static void main (String[] args) {
        int[] âges = générerTableau(120, 1000000);
        int[] comptage = répartitionÂges(âges, 120);
        âges = triÉnumération(âges, comptage);

        System.out.println("Âges triés: " + Arrays.toString(âges));

        // Calcul du nombre de mineurs
        int nMineurs = 0;
        for (int i=0; i < 18; i++){
            nMineurs += comptage[i];
        }
        System.out.println("Nombre de mineurs: " + nMineurs);

        int nCentenaires = 0;
        for (int i=100; i <= 120; i++){
            nCentenaires += comptage[i];
        }
        System.out.println("Nombre de centenaires: " + nCentenaires);

        int nRien = 1000000 - nCentenaires - nMineurs;
        System.out.println("Nombre de personnes ni majeures ni centenaires: "
                + nRien);
    }
}
```

```java
package pbt.evaluation.dev1.janvier;

import org.junit.Test;
import org.junit.Before;
import java.util.Arrays;
import static org.junit.Assert.*;
import static pbt.evaluation.dev1.janvier.Population.*;


public class TestPopulation {

    private int[] âges;
    private int âgeMaximum;

    @Before
    public void initialize(){
        âgeMaximum = 5;
    }

    @Test
    public void répartitionÂges_1(){
        // test cas limite avec un tableau vide
        âges = new int[0];
        int[] résultatAttendu = new int[âgeMaximum + 1];
        assertTrue(
                Arrays.equals(répartitionÂges(âges, âgeMaximum),
                    résultatAttendu));
    }

    @Test
    public void répartitionÂges_2(){
        // test cas général
        âges = new int[] {1,2,2,3,3,3,4,5};
        int[] résultatAttendu = new int[] {0, 1, 2, 3, 1, 1};
        assertTrue(
                Arrays.equals(répartitionÂges(âges, âgeMaximum),
                    résultatAttendu));
    }

    @Test
    public void répartitionÂges_3(){
        // test cas avec toutes les mêmes valeurs
        âges = new int[] {4,4,4,4,4,4,4,4};
        int[] résultatAttendu = new int[] {0, 0, 0, 0, 8, 0};
        assertTrue(
                Arrays.equals(répartitionÂges(âges, âgeMaximum),
                    résultatAttendu));
    }
    @Test
    public void répartitionÂges_4(){
        // test cas changement de l'âge maximal
        âges = new int[] {0,0};
        âgeMaximum = 0;
        int[] résultatAttendu = new int[] {2};
        assertTrue(
                Arrays.equals(répartitionÂges(âges, âgeMaximum),
                    résultatAttendu));
    }
    @Test(expected=IllegalArgumentException.class)
    public void répartitionÂges_5(){
        // test tableau null
        âges = null;
        répartitionÂges(âges, âgeMaximum);
    }
}
```