ALG2 - Lucky Summary

Sm!le42

12 avril 2021

Table des matières

1 Notations **1**
- 1.1 Méthode . . . 1
- 1.2 Print . . . 1
- 1.3 Variable . . . 1
- 1.4 Condition (If) . . . 2
- 1.5 Boucles (For, For Each, While) . . . 2
  - 1.5.1 For . . . 2
  - 1.5.2 For Each . . . 2
  - 1.5.3 While . . . 2
- 1.6 Classe . . . 3

2 Exercices **3**
- 2.1 Ruptures . . . 3
  - 2.1.1 La chasse au gaspi . . . 3
  - 2.1.2 Compter les étudiants . . . 3
  - 2.1.3 Les fanas d'info . . . 6
  - 2.1.4 Éliminer les doublons . . . 7

1 Notations

1.1 Méthode

algorithm *nomMéthode(paramètres)* **—> *retour***
  Instructions...
  return *retour*
end

Algorithme 1 : Méthode

1.2 Print

```
/* Affiche "Hello World!" à l'écran                                    */
algorithm testPrint()
  print "Hello World !"
end
```

Algorithme 2 : Print

1.3 Variable

```
/* Créé une variable newVar qui vaut 7                                 */
/* Ajoute la valeur de test donné en paramètre à la variable newVar    */
/* Définit la quatrième valeur du tableau de booléens tab à true       */
/* Retourne le String "OK"                                             */
algorithm testVariables(test : integer, tab : array of n × m boolean) —> String
  int newVar = 7
  newVar += test
  tab[3] = true
  return "Ok"
end
```

Algorithme 3 : Variable

1.4 Condition (If)

```
/* Retourne le signe de l'entier test donné en paramètre                    */
algorithm testConditions(test : integer) —> String
    String result = "error"
    if test < 0 then
        result = "negative"
    else
        result = "positive"
    end
    return result
end
```

Algorithme 4 : Condition

1.5 Boucles (For, For Each, While)

1.5.1 For

```
/* Vérifie si le tableau d'objets donné en paramètre est vide               */
/* Retourne true si le tableau est vide, false si il y a au moins 1 élément */
algorithm testFor(tab : array of n × m objects) —> boolean
    for i from 0 to tab.length[0] do
        for j from 0 to tab.length do
            if tab[i][j] != null then
                return false
            end
        end
    end
    return true
end
```

Algorithme 5 : For

1.5.2 For Each

```
/* Vérifie si tous les entiers de la liste donnée en paramètre sont positifs */
/* Retourne true tous positifs, false si il y a au moins 1 négatif           */
algorithm testForEach(list : arrayList of n integers) —> boolean
    foreach int nb : tab do
        if nb < 0 then
            return false
        end
    end
end
return true
```

Algorithme 6 : For Each

1.5.3 While

```
/* Vérifie si le tableau d'objects donné en paramètre est vide              */
/* Retourne true si le tableau est vide, false si il y a au moins 1 élément */
algorithm testWhile(array of n objects) —> boolean
    int i = 0
    while i < tab.length and tab[i] == null do
        i++
    end
    return i == tab.length
end
```

Algorithme 7 : While

1.6 Classe

```
/* Exemple de classe Contact comprenant deux attributs privés Nom et ID        */
/* Avec une méthode constructor pour créer une instance                        */
/* Ainsi que deux getter pour pouvoir accéder aux attributs                    */
/* Et enfin, deux méthodes permettant de supprimer ou modifier un contact      */
class Contact
    private :
        String name
        int ID
    public :
        constructor Contact(String name)

        method getName( ) —> String
        method getID( ) —> int

        method deleteContact(int ID)
        method changeName(String newName)
end
```

Algorithme 8 : Classe

2 Exercices

2.1 Ruptures

2.1.1 La chasse au gaspi

À l'ÉSI, les quantités de feuilles imprimées et photocopiées par les professeurs et les étudiants sont enregistrées à des fins de traitement. Le service technique désirant facturer les « exagérations », vous fournit une liste de toutes les impressions effectuées depuis le début de l'année. Cette liste est composée d'objets de la classe Job suivante et est ordonnée alphabétiquement en majeur sur le champ login :

```
class Job
    private :
        String login
        date date
        int nombre
    public :
        Constructeur et getteur
end
```

Algorithme 9 : La chasse au gaspi - énnoncé

Écrire un algorithme permettant d'afficher une ligne (avec login et nombre) par utilisateur dont le nombre total de feuilles imprimées dépasse une valeur limite entrée en paramètre.

Solution non vérifiée pas un enseignant... Attention aux erreurs !

```
algorithm tooManyCopies(int limit, ArrayList<Job> list)
    foreach Job job : list do
        if job.getNombre() > limit then
            println "Login : " + job.getLogin() + " | Nombre : " + job.getNombre()
        end
    end
end
```

Algorithme 10 : La chasse au gaspi - solution

2.1.2 Compter les étudiants

Supposons que la classe `Etudiant` contienne également un attribut indiquant dans quel bloc se trouve l'étudiant (1, 2 ou 3). On voudrait un algorithme qui reçoit une liste d'étudiants et calcule le nombre d'étudiants dans chaque section et, par section, dans chaque bloc.

L'affichage ressemblera à :

```
Gestion
    bloc 1: 130 étudiants
    bloc 1: 42 étudiants
    bloc 1: 16 étudiants
    Total: 188 étudiants
```

```
Industriel
     bloc 1: 32 étudiants
     bloc 2: 14 étudiants
     bloc 3: 8 étudiants
     Total: 54 étudiants
Réseau
     bloc 1: 82 étudiants
     bloc 2: 31 étudiants
     bloc 3: 13 étudiants
     Total: 126 étudiants
```

1. Quel doit-être le tri pour pouvoir résoudre cet exercice avec un algorithme de rupture ?
2. Écrire cet algorithme

Solution non vérifiée pas un enseignant... Attention aux erreurs !

1. Rupture de niveau 2.
2.

```
algorithm countStudents(Etudiant students)
    int[] result = {0, 0, 0, 0, 0, 0, 0, 0, 0}
    foreach Etudiant student : students do
        switch student.getSection() do
            case "G" do
                switch student.getBloc() do
                    case 1 do
                        result[0]++
                    end
                    case 2 do
                        result[1]++
                    end
                    case 3 do
                        result[2]++
                    end
                end
            end
            case "I" do
                switch student.getBloc() do
                    case 1 do
                        result[3]++
                    end
                    case 2 do
                        result[4]++
                    end
                    case 3 do
                        result[5]++
                    end
                end
            end
            case "R" do
                switch student.getBloc() do
                    case 1 do
                        result[6]++
                    end
                    case 2 do
                        result[7]++
                    end
                    case 3 do
                        result[8]++
                    end
                end
            end
        end
    end
    println "Gestion"
    println "bloc1 : " + result[0]
    println "bloc2 : " + result[1]
    println "bloc3 : " + result[2]
    println "Total : " + (result[0] + result[1] + result[2])
    println "Industriel"
    println "bloc1 : " + result[3]
    println "bloc2 : " + result[4]
    println "bloc3 : " + result[5]
    println "Total : " + (result[3] + result[4] + result[5])
    println "Réseau"
    println "bloc1 : " + result[6]
    println "bloc2 : " + result[7]
    println "bloc3 : " + result[8]
    println "Total : " + (result[6] + result[7] + result[8])
end
```

Algorithme 11 : Compter les étudiants - solution

2.1.3 Les fanas d'info

Une grande société d'informatique a organisé durant les douze derniers mois une multitude de concours ouverts aux membres de clubs d'informatique. Elle souhaiterait récompenser le club qui aura été le plus « méritant » durant cette période au point de vue de la participation des membres mineurs. Chaque résultat individuel des participants (y compris des majeurs) est repris dans une liste dont les éléments sont de type Participant.

```
class Participant
    private :
        String nom
        int age
        String reference
        int numero
        int resultat
    public :
        Constructor et getteurs
end
```

Algorithme 12 : Les fanas d'info - énnoncé

Sachant que la liste est ordonnée en majeur sur le champ référence et en mineur sur le champ nom, on demande d'écrire l'algorithme qui affiche les informations suivantes :

pour chaque club :
- sa référence

pour chaque membre mineur de ce club :
- son nom et prénom
- la cote moyenne sur 100 des concours auquel ce membre a participé
- le nombre total de participations des membres mineurs

N.B. : un membre mineur qui s'est inscrit à un concours = une participation. Un club qui n'aura eu aucun membre mineur participant figurera quand même dans le résultat avec la mention "Pas de participation de membre mineur". Par contre, un club dont aucun membre n'a participé au moindre concours ne sera pas affiché. À la fin, on affichera la référence du meilleur club, à savoir celui qui a eu la plus haute cote moyenne de membres mineurs (simplifions en ne gérant pas les possibles ex-æquo).

Solution non vérifiée pas un enseignant... Attention aux erreurs !

```
algorithm getMinorStats(ArrayList<Participant> participants)
    int cpt = 0/* Compteur total des éléments de la liste                    */
    String tempRef/* Référence temporaire (du club)                          */
    String tempName/* Nom temporaire (du participant)                        */
    int nbPart = 0/* Nombre de participants (du club)                        */
    int nbCompet = 0/* Nombre de compétitions (du participant)               */
    int totalPoints = 0/* Total de points (du participant)                   */
    while cpt < participants.size() do
        println "Club : " + participants.get(cpt).getReference()
        tempRef = participants.get(cpt).getReference()
        while participants.get(cpt).getReference().equals(tempRef) do
            if participants.get(cpt).getAge() < 18 then
                println "Name : " + participants.get(cpt).getNom()
                tempName = participants.get(cpt).getName()
                nbCompet = 0
                totalPoints = 0
                npPart ++
                while participants.get(cpt).getName().equals(tempName) do
                    nbCompet++
                    totalPoints += participants.get(cpt).getResultat()
                    cpt++
                end
                println "Cote moyenne sur 100 : " + (totalPoints/nbCompet) nbCompet = 0
                totalPoints = 0
            else
                cpt++
            end
        end
        println "Nombre total de participants mineurs : " + nbPart
        nbPart = 0
    end
end
```

Algorithme 13 : Les fanas d'info - solution

2.1.4 Éliminer les doublons

Soit une liste ordonnée d'entiers avec des possibles redondances. Écrire un algorithme qui enlève les redondances de la liste. On vous demande de créer une nouvelle liste (la liste de départ reste inchangée).

(Ex : Si la liste est {1, 3, 3, 7, 8, 8, 8} alors le résultat sera {1, 3, 7, 8})

Solution :

```
algorithm delDuplicates(ArrayList<Integers> list) —> ArrayList<Integers>
    ArrayList<Integers> result = new ArrayList<>
    result.add(list.get(0))
    int tempNb = list.get(0)
    foreach int nb : list do
        if nb != tempNb then
            result.add(nb)
        end
        tempNb = nb
    end
    return result
end
```

Algorithme 14 : Éliminer les doublons - solution

Test du code :

```java
import java.util.*;
public class Ruptures5 {
    public static void main(String[] args) {
        ArrayList<Integer> test = new ArrayList<>();
        test.add(1);
        test.add(3);
```

```java
        test.add(3);
        test.add(7);
        test.add(8);
        test.add(8);
        test.add(8);
        System.out.println("<Exécution du code>");
        System.out.println("Liste de base: " + test);
        ArrayList<Integer> result = delDuplicates(test);
        System.out.println("Liste modifiée: " + result);
        System.out.println("L'algorithme fonctionne!");
    }

    public static ArrayList<Integer> delDuplicates(ArrayList<Integer> list) {
        ArrayList<Integer> result = new ArrayList<>();
        result.add(list.get(0));
        int tempNb = list.get(0);
        for(int nb : list) {
            if (nb != tempNb) {
                result.add(nb);
            }
            tempNb = nb;
        }
        return result;
    }
}
```

```
<Exécution du code>
Liste de base: [1, 3, 3, 7, 8, 8, 8]
Liste modifiée: [1, 3, 7, 8]
L'algorithme fonctionne!
```