

## Initiation à l'algorithmique

### Remarques préliminaires :

Les notes de TD et TP sont autorisées.

Le devoir de 5 pages comporte 3 exercices et un problème indépendants.

Les durées sont indicatives.

### Exercice 1 (15 mn)

On considère la fonction suite qui calcule le terme de rang n d'une suite :

```
static int suite (int u0, int n){
    int terme = u0;
    for(int i = 1; i <= n ; i++){
        if ((terme % 2) == 0){//terme pair ?
            terme = terme / 2;
        }else{
            terme = 3*terme + 1;
        }//else
        System.out.println(terme);
    }//for
    return terme;
}//suite
```

#### Question 1

Donner les valeurs successives de la variable terme affichées à chaque pas de la boucle for pour l'appel

suite(5, 8)

Quelle est la valeur rendue par la fonction ?

#### Question 2

Pour cette suite, il y a une conjecture qui dit que

$$(\forall u_0) (\exists n) \quad u_n = 1$$

Ecrire une fonction rang qui pour  $u_0$  donné, calcule la plus petite valeur de n pour laquelle  $u_n = 1$ .

```
static int rang(int u0)
```

## Exercice 2 (20mn)

On considère un tableau d'entiers de longueur impaire contenant des valeurs toutes distinctes.

Un entier  $m$  appartenant au tableau est médiane de l'ensemble des valeurs si l'ensemble contient autant de valeurs inférieures que de valeurs supérieures à  $m$ .

Soit  $t$  le tableau ci-dessous

0	1	2	3	4	5	6
3	9	11	2	6	1	7

6 est médiane de l'ensemble des valeurs : il a 3 valeurs inférieures 3, 2, 1 et 3 valeurs supérieures 9, 11, 7

### Question 1

Ecrire une fonction `estMediane` qui vérifie qu'un entier est médiane d'un tableau :

```
static boolean estMediane(int [] tab, int m)
```

### Question 2

Extraire d'un tableau le sous-ensemble des valeurs inférieures à la médiane :

```
static int[] tabInf(int[] tab, int m)
```

Sur l'exemple, l'appel `tabInf(t,6)` délivrera [3, 2, 1]

## Exercice 3 (25 mn)

On se place dans le cadre du TP4 sur le choix des options. Pour vous aider à comprendre les questions voici un exemple de données :

```
int[] placesTot = {3, 2, 4, 3, 2, 2};
int [][] tabChoix = {
    {0, 1, 2, 3, 4, 5},
    {1, 3, 4, 0, 2, 5},
    {1, 5, 4, 3, 2, 0},
    {4, 3, 2, 5, 0, 1},
    {2, 3, 4, 5, 1, 0},
    {0, 1, 2, 3, 4, 5},
    {4, 5, 0, 1, 2, 3},
    {5, 4, 3, 2, 1, 0},
    {3, 2, 1, 0, 5, 4}
}
```

### Question 1

Ecrire une fonction qui calcule le nombre de demandes en 1<sup>er</sup> choix pour chaque option :

```
static int[] premiersChoix(int [][] tChoix)
```

qui sur l'exemple délivrera

0	1	2	3	4	5
2	2	1	1	2	1

### Question 2

Ecrire une fonction qui vérifie que tous les étudiants pourront avoir leur premier choix connaissant le nombre de demandes en premier choix pour chaque option et le nombre de places totales dans l'option :

```
static boolean tousSatisfaits(int[] premChoix, int [] placesTot)
```

### Question 3

Ecrire une fonction qui calcule le nombre d'étudiants à recruter à l'extérieur pour chaque option sachant que tous les étudiants ont eu leur premier choix :

```
static int[] nbExternes (int [] premChoix, int [] placesTot)
```

## Problème : Outils pour la gestion de plannings (1h)

Une société savante veut organiser un congrès qui se déroule sur une journée de 8h à 20h. Tous les participants assistent à la séance d'ouverture de 8h à 9h et à la séance de clôture de 19h à 20h. Le reste de la journée se passe en ateliers qui débutent et se terminent sur des frontières d'heures. Il y a plusieurs ateliers en parallèles. Chaque participant choisit les ateliers qui l'intéressent en tenant compte des horaires fixés. L'objectif du problème est de définir des outils pour aider à la gestion du planning de la journée pour chaque participant.

On définit tout d'abord une classe `Creneau` qui permet de modéliser un atelier qui se déroule sur un créneau horaire dont on connaît l'heure de début et l'heure de fin par exemple (9h-11h). On pourrait y ajouter logiquement le titre de l'atelier mais on choisit de l'ignorer vu qu'il n'intervient pas dans la suite de notre problème.

On définit ensuite une classe `Planning` qui modélise le planning de la journée d'un participant. Un planning est un ensemble ordonné de créneaux par exemple {(8h-9h) (9h-11h) (12h-14h) (15h-18h) (19h-20h)}.

Le problème consiste à compléter les deux classes `Creneau` et `Planning` dont on vous donne les squelettes. Les méthodes `toString()` sont données pour justifier l'affichage des résultats des méthodes `main`. Les résultats affichés sont mis en commentaire aussitôt après le `println` ayant provoqué l'impression. Un commentaire indique le rôle de chaque méthode à compléter. Lors de la rédaction, recopier l'en-tête de chaque méthode sur votre copie.

Pensez à utiliser les méthodes qui ont été définies. Vous pouvez, si vous y voyez un intérêt, définir de nouvelles méthodes.

Pour se fixer les idées, voici quelques schémas représentant un objet `Planning` au cours des ajouts successifs de créneaux indiqués dans la méthode `main` de la classe `Planning`.

Schéma 1 : l'objet `Planning p` aussitôt après l'appel du constructeur

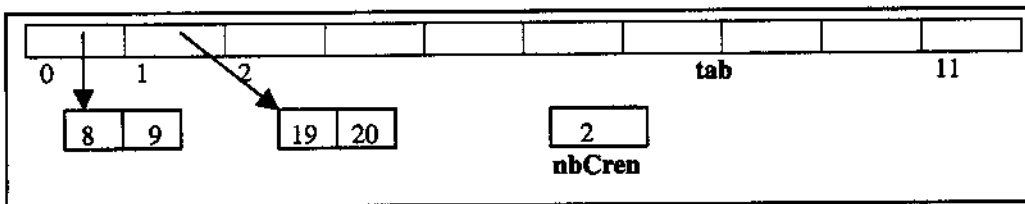


Schéma 2 : l'objet `Planning p` après insertion du premier créneau c11A12.

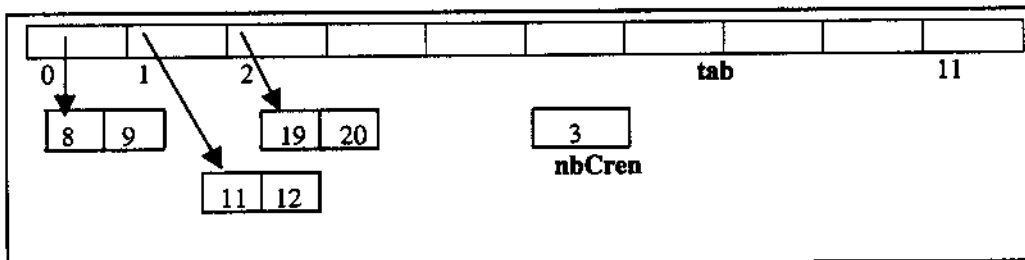
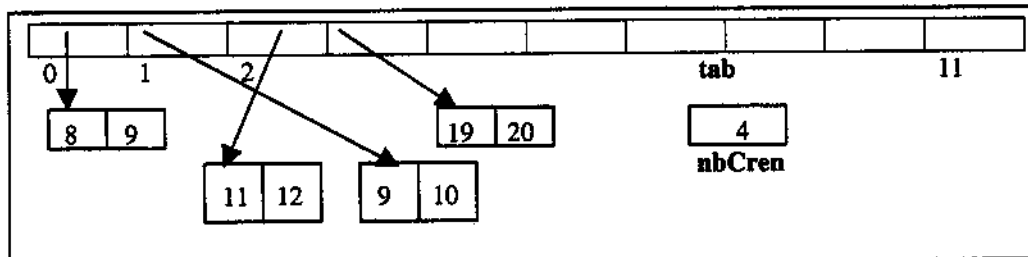


Schéma 3 : l'objet `Planning p` après insertion de c9A12 (refusée) et c9A10 acceptée :



```
public class Creneau{
```

4

## La classe Planning

```
public class Planning{
//les attributs
    Creneau [] tab;// c'est un tableau ordonné dans l'ordre croissant
                    // des débuts de créneaux
    int nbCren; // le nombre de créneaux effectifs

    public Planning(){//le constructeur
        tab=new Creneau [12]; // au maximum 12
        tab[0]=new Creneau(8,9);
        tab[1]=new Creneau(19,20);
        // tous les participants sont présents de 8 à 9 et de 19 à 20
        // tous les autres créneaux ajoutés seront compris entre 9h et 19h
        nbCren=2;
    }//constructeur Planning

    public boolean correct( ){// à compléter
        //vérifie que le planning est correct =>
        //ordre croissant et pas de chevauchement d'intervalles
        //on suppose les créneaux bien construits
    }//correct

    public int place(Creneau c){ // à compléter
        // délivre l'indice i du tableau où doit
        // se placer le Creneau c ou -1 si impossible
        // le Creneau c peut s'insérer entre le Creneau tab[i-1] et le
        // Creneau tab[i]
    }//place

    public boolean insere (Creneau c){ // à compléter
        //insere le Creneau c après avoir cherché sa place en décalant tous
        //les autres créneaux situés après dans le tableau
        //rend vrai si possible et faux sinon
    }//insere

    public Creneau libre(int d){ // à compléter
        // recherche un Creneau disponible pour la durée d nécessaire
        //il existe un "trou" d'une durée>=d entre tab[i] et tab[i+1]
        //délivre le Creneau [h,h+d] où h est l'heure de fin
        //du Creneau tab[i] ; rend le Creneau(0,0) si impossible
    }//libre

    public Planning creneauxLibres(){ question subsidiaire à compléter
        //calcule le Planning obtenu en extrayant les créneaux libres du
        // Planning en référence
    }//creneauxLibres

    public String toString(){
        String resul ="(";
        for(int i=0;i<nbCren;i++){
            resul = resul +tab[i]+" ";
        }
        return resul+")";
    }//toString

    public static void main(String [] args){
        Creneau c9A10=new Creneau(9,10);
        Creneau c9A12=new Creneau(9,12);
        Creneau c11A12=new Creneau(11,12);
        Planning p=new Planning();//contient déjà les créneaux 8hà9h et 19hà20h
        System.out.println(p.insere(c11A12)+" "+p);
            // true {(8h-9h) (11h-12h) (19h-20h) }
        System.out.println(p.insere(c9A12)+" "+p);
            //false {(8h-9h) (11h-12h) (19h-20h) }
        System.out.println(p.insere(c9A10)+" "+p);
            //true {(8h-9h) (9h-10h) (11h-12h) (19h-20h) }
        System.out.println("creneau libre "+p.libre(2));
            //libre (12h-14h)
    }//main
}//class Planning
```