

Seul document autorisé : une feuille A4 recto-verso manuscrite (non photocopiée). Aucune machine.
Le barème est indicatif. Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.
L'utilisation de collections de l'API Java (en particulier `LinkedList` et `ArrayList`) est interdite.

- Tous les attributs d'instance de toutes les classes introduites doivent être privés.
- Il est possible d'écrire des méthodes non spécifiquement demandées si c'est plus pratique, en spécifiant pour chaque méthode dans quelle classe elle se trouve.

Exercice 1. (1 point) Rappeler brièvement (on ne demande pas nécessairement de code) la stratégie utilisée en cours pour implémenter un tableau de taille variable et expliquer les inconvénients de cette stratégie.

Le but de ce sujet est d'écrire une implémentation de tableau de taille variable qui ne présente pas les inconvénients de l'implémentation vue en cours. On prend comme prétexte la gestion d'une liste de clients d'un magasin.

1 Client

Un client est représenté par son nom et son prénom (peu importe les doublons pour cause d'homonymie). Toutes les cent inscriptions, le client qui s'inscrit reçoit un cadeau de bienvenue. On modélise ainsi un client comme une instance de la classe `Client` qui contient alors au moins deux attributs `nom` et `prenom`, de type chaîne de caractères et qui admet l'interface :

```
/** renvoie une presentation avec les nom et prenom du client */
String toString();
```

Cette classe possède un constructeur qui met en place les attributs `nom` et `prenom`, et qui affiche, le cas échéant, le message "remettre un cadeau de bienvenue".

Exercice 2. (1,5 points) Écrire la classe `Client` (attributs, constructeurs et méthodes).

2 Tableau de taille variable

On représente un tableau de taille variable de `clients` par un arbre d'arité constante 10 (c'est-à-dire tel qu'un nœud qui n'est pas une feuille possède 10 fils). Chaque nœud de l'arbre possède :

- un attribut qui permet de stocker un `Client` (si c'est une feuille) et
- un attribut qui permet de stocker ses 10 fils (si ce n'est pas une feuille).

La hauteur de l'arbre dépend du nombre de cases utilisées du tableau de taille variable qu'on est en train d'implémenter, plus précisément du nombre de chiffres de son plus grand indice (en base 10). Des exemples sont donnés en figure 1.

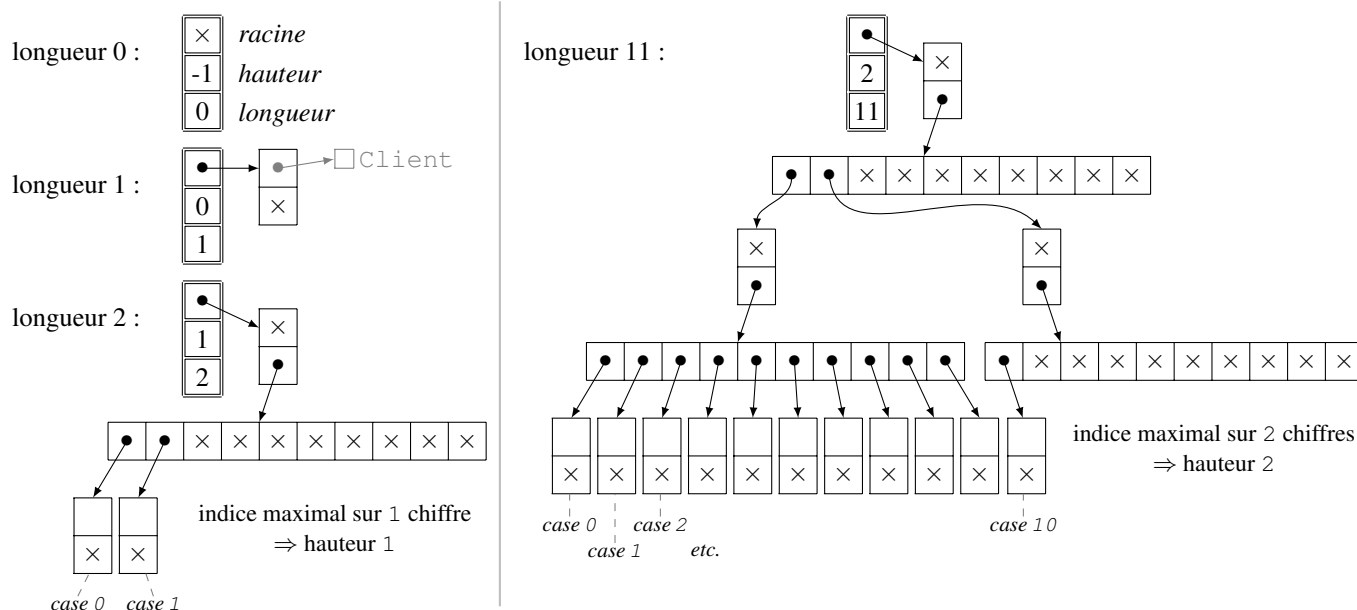


FIGURE 1 – Exemples de représentation d'un tableau de taille variable de `clients` par un arbre. La référence vers le client n'est représentée que pour la longueur 1. Le symbole `x` désigne `null`, `•` une instance de `Noeud` et `⊠` une instance de `Tableau`.

On utilise deux classes pour représenter un tableau de taille variable sous forme d'un arbre : `Tableau` qui référence la `racine` et possède des attributs `longueur` (=nombre de cases utilisées dans le tableau de taille variable) et `hauteur` (de l'arbre), et `Noeud` qui représente un nœud de l'arbre.

Exercice 3. (1 point) Écrire les déclarations des attributs des classes **Tableau** et **Noeud**.

L'interface de la classe **Tableau** est la suivante :

```
/** ajoute un client a la fin du tableau de taille variable (->creation d'une nouvelle case) */
void addLast(Client c);

/** renvoie le client de la case d'indice k, null s'il n'y en a pas */
Client get(int k);

/** renvoie l'indice d'une case contenant le parametre, -1 s'il n'apparait pas */
int contains(Client c);

/** modifie le client de la case d'indice k si elle est dans les limites du tableau de taille variable,
renvoie true si et seulement si la modification a eu lieu */
boolean set(int k, Client c);
```

Exercice 4. (1 point) Pour accéder à une case d'un tableau de taille variable, il faut parcourir l'arbre le représentant. La première case d'un tel tableau est indiquée par 0.

1. Dessiner l'arbre représentant un tableau de taille variable à 132 cases (on pourra utiliser des pointillés) et indiquer les nœuds correspondant aux cases d'indices respectifs 8, 18 et 108.
2. Donner une stratégie pour atteindre le nœud correspondant à une case à partir de son indice et de la hauteur de l'arbre (on ne demande pas de code).

Pour la suite, on fournit une classe **Entier** d'interface :

```
/** renvoie le nombre correspondant aux n derniers chiffres (ie les plus a droite)
ex: NDerniersChiffres(1234, 3) renvoie 234 */
static int NDerniersChiffres(int k, int n);

/** renvoie le chiffre en position donnee (numeration de droite a gauche a partir de 1)
ex: chiffreEnPos(1234, 1) renvoie 4; chiffreEnPos(1234, 5) renvoie 0 */
static int chiffreEnPos(int k, int pos);

/** teste si un nombre est une puissance de 10 */
static boolean estPuissanceDeDix(int k);
```

Exercice 5. (5 points) Écrire les méthodes **get**, **contains** et **set** de la classe **Tableau**. Pour la méthode **contains**, on demande une solution récursive sur l'arbre représentant le tableau de taille variable.

Exercice 6. (3 points) Écrire la méthode **addLast** de la classe **Tableau**. Dans le cas où la capacité du tableau augmente, votre implémentation doit suivre la stratégie illustrée en figure 2 : en particulier aucune partie de l'arbre ne doit être recopiée.

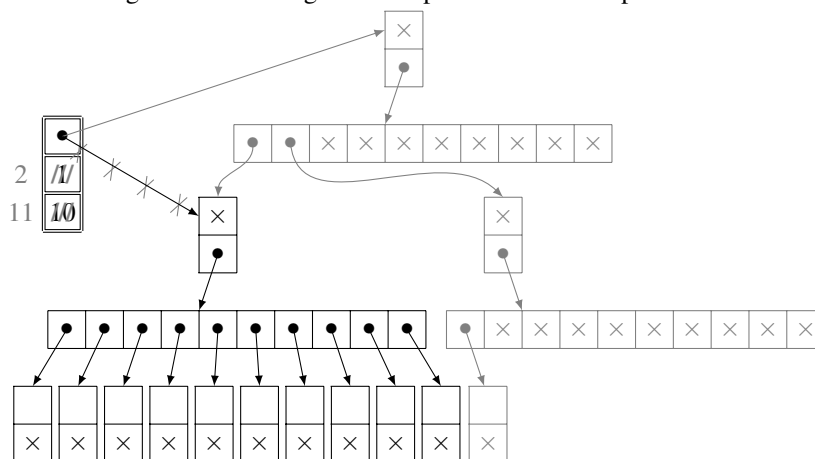


FIGURE 2 – Augmentation de capacité – création d'une onzième case : ce qui est créé ou modifié est représenté en gris.

3 Suppression d'éléments

On veut maintenant pouvoir supprimer des éléments d'un tableau de taille variable. **On ne demande pas de mettre à jour les méthodes précédentes pour en tenir compte.**

Exercice 7. (2 points) Implémenter la méthode suivante dans la classe **Tableau** :

```
/** passe a null l'attribut de type Client de la case correspondante et renvoie son ancienne valeur */
Noeud unset(int k);
```

Pour éviter de perdre de la place, on souhaite maintenir à jour une liste de **Noeuds** libres et donc réutilisables.

Exercice 8. (4 points) Mettre en place les classes permettant de gérer une liste de **Noeuds** premier entré / dernier sorti (attributs, constructeurs, méthodes).

Exercice 9. (1,5 points) Écrire une méthode **add(Client c)** dans la classe **Tableau** qui ajoute un client au tableau de taille variable : dans une case non occupée s'il y en a, à la fin sinon.