Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

TD n°7

Objectifs: Apprendre à utiliser les tableaux.

ATTENTION ! Il faut être attentif aux modes de passage de paramètres. Tous les types primitifs (numériques, caractères, booléen) sont passés **par valeur** (ie. c'est une copie qui est transmise à la fonction) alors que les types tableaux sont passés **par référence** (c'est l'adresse en mémoire du tableau qui est copiée !). Ainsi, si vous passez en paramètre un tableau à une fonction et que cette dernière le modifie, une fois revenu dans le programme principal le tableau aura été modifié (puisqu'il n'y a pas eu de copie).

Rappel sur les tableaux

Un tableau en `iJava` se déclare comme une variable quelconque en ajoutant juste des crochets juste derrière le nom du type `<type>[]`.

Exemple: `int[] notesAlgo;`

Pour allouer un tableau, on utilise une affectation en précisant le nombre de cases que l'on désire utiliser, leur type et le mot-clé `new` pour allouer la mémoire nécessaire.

Exemple: `notesAlgo = new int[5]`, si l'on désire travailler avec un tableau de 5 cases.

Pour accéder à une des valeurs contenues dans le tableau, il suffit de spécifier son indice, que cela soit pour la lire ou la modifier.

Exemple: `notesAlgo[0] = notesAlgo[1]`, pour copier la valeur située dans la case d'indice 1 dans celle d'indice 0.

Pour obtenir le nombre de cases que contient un tableau, il faut utiliser la fonction `length`, dont voici la définition: `int length(<type>[] t)`. Si le tableau n'est pas encore crée/alloué, une erreur se produit !

Exemple: `length(notesAlgo)` retournera `5`.

Exercice 1 : Créer ses premiers tableaux

Dans cet exercice vous allez afficher les valeurs contenues dans un tableau, et créer un tableau.

Soit `tab` le tableau ci-dessous. Les nombres variant de 10 à 100 correspondent au contenu des cases du tableau, tandis que les indices repérant chacune des cases du tableau apparaissent en dessous (en italique).

valeurs	10	20	30	40	50	60	70	80	90	100
indices	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

1. En supposant que le tableau `tab` est déclaré et alloué, écrire une boucle à compteur qui affiche les nombres contenus dans le tableau séparés par des espaces. On affichera les valeurs de gauche à droite.
2. En supposant à nouveau que `tab` est déclaré et alloué, écrire une boucle à compteur qui affiche les valeurs contenus dans `tab` dans l'ordre de droite à gauche.
3. Écrire l'algorithme `void algorihtm()` qui déclare et alloue le tableau `tab`, puis utilise une boucle à compteur pour l'initialiser avec les valeurs comme ci-dessus.
4. Modifier l'algorithme de la question précédente en utilisant une déclaration en extension, qui déclare, alloue et initialise le tableau dans une seule instruction.
5. Écrire une fonction `int[] creer(int tailleTableau)` qui crée et retourne un tableau similaire à `tab`, mais dont la taille n'est pas fixée à 10 mais donnée par le paramètre `tailleTableau`. La fonction doit passer ce test

```
1 void testCreer() {
2   assertArrayEquals(new int[]{10, 20, 30}, creer(3));
3   assertArrayEquals(new int[0], creer(0));
4 }
```

6. Est-ce qu'on peut écrire la fonction `int[] creer(int tailleTableau)` de la question précédente en utilisant une déclaration par extension au lieu d'utiliser une boucle ?

Exercice 2 : Calculer la moyenne des valeurs d'un tableau

On désire calculer la moyenne des valeurs contenues dans un tableau d'entiers. On supposera que le tableau contient au moins une valeur (autrement la moyenne n'est pas définie).

Vous devez écrire la fonction `int moyenne(int[] tab)`, en utilisant une boucle à compteur. La fonction doit passer ce test

```
void testMoyenne() {
  assertEquals(5, moyenne(new int[]{5}));
  assertEquals(2, moyenne(new int[]{1,2,3}));
  assertEquals(10, moyenne(new int[]{5,15}));
}
```

Exercice 3 : Recherche séquentielle d'une valeur dans un tableau

Nous allons dans cet exercice nous intéresser à la recherche d'une valeur dans un tableau d'entiers. La *recherche séquentielle* consiste à visiter toutes les valeurs du tableau dans l'ordre et les comparer avec la valeur recherchée jusque ce que la valeur est trouvée ou le tableau est épuisé.

1. Dans une première version plus simple, nous voulons simplement vérifier si la valeur est présente. Écrire la fonction `boolean estPresente(int[] tab, int valeur)` qui retourne `true` si `valeur` est présente dans `tab`, et retourne `false` sinon. Vous devez vous assurer que la boucle s'arrête dès que la valeur est trouvée. La fonction doit passer ce test

```
void testEstPresent() {
  assertTrue(estPresente(new int[]{1,3,5,7,9}, 5));
  assertTrue(estPresente(new int[]{4,5,5,5}, 5));
  assertFalse(estPresente(new int[]{1,3,5,7,9}, 2));
  assertFalse(estPresente(new int[0], 10));
}
```

2. Dans une deuxième version plus complexe, on voudrait non seulement vérifier si la valeur est présente, mais aussi retourner l'indice où elle se trouve. Écrire la fonction `int indiceDe(int[] tab, int valeur)` qui retourne l'indice de la *première* occurrence de `valeur` dans `tab`, et retourne -1 si la valeur n'est pas présente. Assurez-vous que la boucle while s'arrête dès que la valeur est trouvée. La fonction doit passer ce test

```
void testIndiceDe() {
  assertEquals(2, indiceDe(new int[]{1,3,5,7,9}, 5));
  assertEquals(1, indiceDe(new int[]{4,5,5,5}, 5));
  assertEquals(-1, indiceDe(new int[]{1,3,5,7,9}, 2));
  assertEquals(-1, indiceDe(new int[0], 10));
}
```

Exercice 4 : Décaler les éléments d'un tableau

On souhaite dans cet exercice réaliser un décalage des éléments d'un tableau.

1. Ecrivez dans un premier temps une fonction `int[] creerTableauOrdonne(int taille)` permettant de créer un tableau initialisé de 1 à $taille$ (càd, que la première case contient la valeur 1, la deuxième la valeur 2, etc ..., la dernière la valeur $taille$).

```
void testCreerTableauOrdonne() {
  assertArrayEquals(new int[]{1,2,3,4,5,6,7,8,9,10}, creerTableauOrdonne(10));
}
```

2. Écrivez une fonction qui permet d'inverser les contenus de la première et la dernière case d'un tableau.

```
void testEchangerDebFin() {
  int[] tab = {59,62}
  echangerDebFin(tab);
  assertArrayEquals(new int[]{62,59}, tab);

  int[] tab2 = new int[]{42};
  echangerDebFin(tab2);
  assertArrayEquals(new int[]{42}, tab2);
}
```

3. Ecrivez ensuite une fonction `void decaler(int[] tab)` permettant de décaler l'ensemble des valeurs du tableau d'une case vers la droite (la dernière valeur devra se retrouver dans la première case !). La fonction doit passer ce test :

```
1 void testDecaler() {
2   int[] tab = new int[]{2, 5, 3, 1, 7};
3   decaler(tab);
4   assertArrayEquals(new int[]{7, 2, 5, 3, 1}, tab);
5
6   int[] tab2 = creerTableauOrdonne(5);
7   decaler(tab2);
8   assertArrayEquals(new int[]{5,1,2,3,4}, tab2);
9 }
```

Prolongements

Exercice 5 : Notes sur 20

On considère un tableau comportant des notes sur 100. Proposez une fonction `sur20` qui retourne un nouveau tableau des notes remises sur 20 et passe le test suivant :

```
void testRemisesSur20(){
  assertArrayEquals(new double[]{12.2, 0.0, 7.4}, sur20(new int[]{61, 0, 37}));
}
```

On se rend compte que certains enseignants adoptent des notations un peu plus atypiques : sur 30, 50 et même 42,42. Généralisez votre fonction pour qu'elle puisse être utilisé quelque soit la valeur de notation de départ.