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

TP n°7

Objectifs: Premiers exercices sur les tableaux.

Quelques éléments de syntaxe sur les tableaux

Nous avons vu en cours notre première structure de données: les tableaux. Cette structure nous permet de manipuler un ensemble de données de même type. Chaque élément de l'ensemble est identifié par un entier appelé indice de l'élément. En `iJava`, l'indice identifiant la première case est 0. On peut représenter un tableau comme une suite de cases identifiées par un indice et pouvant contenir une variable d'un certain type. L'exemple ci-dessous illustre un tableau contenant 4 entiers:

4	7	-1	3
0	1	2	3

Comme vous le remarquez, il faut bien distinguer les valeurs contenues dans les cases du tableau (ie. les variables) de l'indice permettant d'identifier une case du tableau en particulier. Avec cette structure de données, il est possible d'accéder directement à la valeur contenue dans une case donnée, juste en précisant son indice.

Nous allons maintenant présenter la syntaxe `iJava` permettant de déclarer un tableau, de l'allouer, d'accéder et modifier la valeur d'une case et finalement d'obtenir la taille du tableau (c'est-à-dire le nombre de cases qu'il contient).

Le tableau ci-dessous synthétise les éléments de syntaxe concernant les tableaux en `iJava`:

	`iJava`
Déclaration	`int[] tab;`
Allocation	`tab = new int[5];`
Déclaration et allocation	`int[] tab = new int[5];`
Accès à une case	`tab[0]`
Modification d'une case	`tab[0] = 3;`
Taille du tableau	`length(tab)`

Exercice 1 : Tableau vers chaine

La fonction `print` de iJava n'affiche pas les valeurs contenues dans un tableau, mais la *référence* (i.e. l'adresse mémoire) de celui ci. Écrire la classe `AffichageTableau` ci-dessous et exécutez la pour voir à quoi ressemble la référence d'un tableau.

```
class AffichageTableau extends Program {
  void algorithm() {
    println(new int[]{1,2,3});
    println(new String[]{"un", "deux", "trois"});
  }
}
```

Or, lorsqu'on programme avec des tableaux, il sera utile de pouvoir afficher leurs valeurs. Dans la suite de l'exercice on voit comment s'y prendre.

1. Dans la classe `AffichageTableau`, ajouter la fonction `String toString(int[] tab)` qui prend en paramètre un tableau d'entiers et retourne une chaîne de caractères qui contient les valeurs de ce tableau séparées par des espaces. La fonction doit passer ce test. (N'oubliez pas de mettre en commentaire la fonction `algorithm` pour que le test soit exécuté.)

```
void testToString_tabInt () {
  assertEquals("1 2 3 ", toString(new int[]{1,2,3}));
  assertEquals("", toString(new int[0]));
}
```

2. Toujours dans la classe `AffichageTableau`, ajouter la fonction `String toString(String[] tab)` qui prend cette fois-ci un tableau de `String` et retourne sa représentation sous forme de chaîne de caractères.

   Remarquez qu'en iJava, on peut dans une même classe définir plusieurs fonctions qui ont le même nom (ici, deux fonctions `toString`), à condition que ces fonctions ont des signatures différentes. Ainsi, le compilateur peut reconnaître laquelle de ces deux fonctions doit être appelées grâce aux arguments d'appel: `toString(new String[]{"un", "deux", "trois"})` appelle la fonction `toString` qui prend en paramètre un tableau de String, tandis que `toString(new int[]{1,2,3})` appelle la fonction `toString` qui prend en paramètre un tableau de int.

   La fonction à écrire doit passer ce test.

```
void testToString_tabString () {
  assertEquals("un␣deux␣trois␣", toString(new String[]{"un", "deux", "trois"}));
  assertEquals("", toString(new String[0]));
}
```

3. Finalement, modifiez la fonction `void algorithm()` pour tester l'affichage produit par vos fonctions `toString`. (N'oubliez pas de décommenter la fonction `algorithm`.)

```
void algorithm() {
  int[] tabInt = new int[]{20, 30, 50};
  String[] tabStr = new String[]{"Alan", "Turing"};

  println(toString(tabInt));
  println(toString(tabStr));
}
```

   Désormais si dans un programme vous avez besoin d'afficher des tableaux, il suffit d'y ajouter une fonction `toString` comme celles définies ici.

Exercice 2 : Manipulations de base pour les tableaux

L'ensemble des fonctions ci-dessous sont à écrire dans le programme `BaseTableaux`.

1. Ecrivez, sans utiliser de déclaration en extension, une fonction `int[] creerTableau()` permettant de créer un tableau d'entiers de 10 cases dont les 5 premières cases contiennent la valeur 1 et les 5 suivantes la valeur 2.

```
1  void testCreerTableau() {
2    assertArrayEquals(new int[]{1,1,1,1,1,2,2,2,2,2}, creerTableau());
3  }
```

2. Que faut-il modifier dans votre fonction pour que l'initialisation se réalise avec une taille de tableau précisée en paramètre (ie. première moitié du tableau ne contenant que des 1 et seconde moitié des 2 ? Écrire cette nouvelle fonction qui prend en paramètre la taille, et qui passe ce test.

```
void testCreerTableau2() {
  assertArrayEquals(new int[]{1,1,1,1,1,2,2,2,2,2}, creerTableau(10));
  assertArrayEquals(new int[]{1,1,2,2,2}, creerTableau(5));
  assertArrayEquals(new int[]{}, creerTableau(0));
}
```

3. Ecrivez une fonction `int[] creerTableauAleatoire(int taille)` qui créée un tableau de *taille* cases contenant des valeurs tirées aléatoirement entre 0 et 20 inclus (utilisez pour cela la fonction de signature `double random()` qui retourne un nombre réel entre `[0.0, 1.0[` et l'instruction de forçage de type `(int)` qui convertit un réel en entier).
4. Définissez la fonction `testCreerTableauAleatoire` qui appelle dans un premier temps la fonction de création d'un tableau aléatoire (`creerTableauAleatoire`) et qui parcourt ensuite chacune des cases en testant l'assertion vérifiant que le contenu de la case est compris entre 0 et 20 inclus (pour cela, utilisez simplement un `assertEquals(true, ...)` prenant en paramètre une expression booléenne devant être vraie).
5. Optionnel Tester l'aléatoire n'est pas facile et le test réalisé ici a d'ailleurs ses limites. Par exemple, si toutes les valeurs de votre tableau sont à 0, le test passerait ... Ajoutez les instructions nécessaires pour que votre test vérifie que toutes les valeurs de 0 à 20 sont bien sorties sur la création d'un tableau de taille 10000 (ça n'est pas certain mais très fortement probable).

Exercice 3 : Vote Majoritaire

On considère rangés dans un tableau les votes sur une proposition. La valeur `true` correspond à un vote pour, la valeur `false` à un vote contre. Pour qu'une proposition soit adoptée, il faut qu'il y ait plus de votes pour que de votes contre. Écrivez le programme `Votons` contenant la fonction de test suivante et la fonction testée.

```
1 void testEstAdopte(){
2   assertEquals(false, estAdopte(new boolean[]{true,false}));
3   assertEquals(true,  estAdopte(new boolean[]{true,true,false}));
4 }
```

Exercice 4 : Valeurs maximale et minimale dans un tableau

Créez un programme `MinMax` dans lequel vous placerez les fonctions déterminant les valeurs maximum et minimum contenues dans un tableau.

1. Concevez une fonction qui retourne la valeur minimale et la valeur maximale contenues dans un tableau de taille $n > 1$. Une fonction ne pouvant retourner qu'une seule valeur, il sera nécessaire d'utiliser un tableau afin de retourner en même temps le mininum et le maximum. Remarquez que l'assertion a légèrement changé afin de vérifier l'égalité du contenu de deux tableaux :

```
1 void testMinMax() {
2   assertArrayEquals(new int[]{1,3}, minMax(new int[]{2,1,3}));
3   assertArrayEquals(new int[]{2,2}, minMax(new int[]{2,2,2}));
4   assertArrayEquals(new int[]{-1,3}, minMax(new int[]{3,-1,2}));
5 }
```

2. Écrire maintenant la fonction `int[] minMaxIndices(int[] tab)` qui retourne les indices de la valeur minimale et de la valeur maximale du tableau `tab`. La fonction `minMaxIndices` doit passer ce test.

```
1 void testMinMaxIndices() {
2   assertArrayEquals(new int[]{1,2}, minMaxIndices(new int[]{2,1,3}));
3   assertArrayEquals(new int[]{0,0}, minMaxIndices(new int[]{2,2,2}));
4   assertArrayEquals(new int[]{1,0}, minMaxIndices(new int[]{3,-1,2}));
5 }
```

Exercice 5 : Le jeu du pendu

Vous connaissez sûrement ce jeu à deux joueurs dans lequel un joueur cherche à deviner (en un nombre de propositions limité) un mot choisi par l'autre joueur. Pour simplifier le problème, nous supposerons dans cet exercice que le mot à deviner est représenté sous la forme d'une chaîne de caractères initialisée en début d'algorithme. On suppose aussi que le premier joueur a le droit à 5 tentatives avant de perdre la partie.

Lorsque la partie débute, le joueur ne voit que le nombre de lettres à deviner, chacune d'elle étant symbolisée par une étoile. A chaque tour, le joueur propose une lettre, si elle est présente dans le mot, l'étoile est remplacée par la lettre correspondante, sinon rien ne se passe (si ce n'est que le joueur perd une tentative). Le joueur gagne s'il découvre le mot avant d'atteindre le nombre d'échecs prévus au début du jeu (5 pour nous).

Voici un exemple de déroulement d'une partie où le joueur perd:

```
 Il vous reste 5 tentatives: * * * * *
Entrez un caractère: a
Il vous reste 4 tentatives: * * * *
Entrez un caractère: e
Il vous reste 3 tentatives: * * * *
Entrez un caractère: n
Il vous reste 3 tentatives: * n * *
Entrez un caractère: o
Il vous reste 2 tentatives: * n * *
Entrez un caractère: i
Il vous reste 2 tentatives: * n i *
Entrez un caractère: a
Il vous reste 1 tentatives: * n i *
Entrez un caractère: s
Vous avez perdu ! Il fallait trouver: unix
```

et une partie où il gagne:

```
 Il vous reste 5 tentatives: * * * *
Entrez un caractère: p
Il vous reste 5 tentatives: p * p *
Entrez un caractère: a
Il vous reste 4 tentatives: p * p *
Entrez un caractère: i
Il vous reste 4 tentatives: p i p *
Entrez un caractère: e
Vous avez gagné ! Il fallait trouver: pipe
```

Maintenant que vous avez une bonne compréhension du jeu, répondez aux questions suivantes avant de vous lancer dans la conception de votre algorithme.

1. Faîtes une partie avec votre voisin(e).
2. Comment procédez-vous pour déterminer les lettres à découvrir au fur et à mesure ? Comment faire pour garder trace des positions déjà découvertes et des positions à représenter par des étoiles ?
3. Quand la partie s'arrête-t-elle ? Quel type de boucle semble le plus approprié ?
4. Concevez maintenant le programme `Pendu` permettant de jouer au jeu du pendu en ne définissant que la fonction `void algorithm()` !
5. *Optionnel* Lorsque vous avez une version fonctionnelle, améliorez là en décomposant votre programme à l'aide de fonctions.

Exercice 6 : Opérations sur des vecteurs

Un vecteur dans l'espace 3D est représenté par trois nombres. Pour simplifier, nous allons considérer que ces nombres sont des entiers. Ainsi, un vecteur sera représenté par un tableau de trois entiers. Dans cet exercice, vous devez écrire plusieurs fonctions de manipulation de vecteurs :

— Saisie d'un vecteur auprès de l'utilisateur.
— Représentation d'un vecteur sous forme de chaîne de caractères.
— Test si deux vecteurs sont égaux.
— Addition de deux vecteurs.
— Produit scalaire de deux vecteurs.

Récupérez sur Moodle le code source `Vecteurs.java` qui contient les tests pour ces fonctions et complétez en ajoutant les définitions des fonctions testées. Il est conseillé de faire les fonctions dans l'ordre d'apparition des tests.

Exercice 7 : Faire l'exercice de prolongement de TD : `NotesSur20.java`