TP 5

LA NOTION DE TABLEAU

yann.secq@univ-lille.fr

Objectifs

- Comprendre et savoir utiliser tableaux à une dimension,
- Création de programmes créant, initialisant et parcourant un tableau.
- Création d'un premier algorithme complexe : le jeu du Mastermind.

Chapitre I : 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)`

Chapitre II : Manipulation de tableau

Exercice 1

Ce premier exercice a pour but de créer et initialiser des tableaux d'entiers.

1. Ecrivez une fonction `void printlnTab(int[] tab)` permettant d'afficher le contenu d'un tableau d'entier en séparant chacun des éléments par un espace.

```
class TP5_Exercice1 extends Program {
  void algorithm(){
    // affiche 1 4 5 2 3
    println1nTab(new int[]{1, 4, 5, 2, 3});
  }
  // ...
}
```

2. Ecrivez maintenant 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.

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

3. 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 ?
4. Ecrivez une fonction `int[] creerTableauAleatoire(int taille)` qui créée un tableau de $taille$ cases contenant des valeurs tirées aléatoirement entre 0 et 20 (utilisez pour cela l'instruction `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).
5. afin de vérifier la validité de votre fonction `creerTableauAleatoire`, définissez une fonction `testCreerTableauAleatoire` qui appelle dans un premier temps la fonction de création d'un tableau aléatoire et qui parcourt ensuite chacune des cases de ce tableau en testant l'assertion vérifiant que le contenu de la case est compris entre 0 et 20 inclus (ie. utilisez un `assertTrue` prenant en paramètre une expression booléenne devant être vraie).

Exercice 2

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

1. Ecrivez dans un premier temps une fonction `int[] creerTableauOrdonne()` permettant de créer un tableau initialisé de 1 à 10 (càd, que la première case contient la valeur 1, la deuxième la valeur 2, etc ...).

```
class TP5_Exercice2 extends Program {
  void testCreerTableauOrdonne() {
    assertArrayEquals(new int[]{1,2,3,4,5,6,7,8,9,10},
      creerTableauOrdonne());
  }
  // ...
}
```

2. Ecrivez ensuite une fonction `void shift(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 !).

```
    void testShift() {
      int[] tab = creerTableauOrdonne();
      shift(tab);
      assertArrayEquals(new int[]{10,1,2,3,4,5,6,7,8,9}, tab);
      tab = new int[]{2, 5, 3, 1, 7};
      shift(tab);
      assertArrayEquals(new int[]{7, 2, 5, 3, 1}, tab);
    }
    // ...
```

Exercice 3

Dans cet exercice, nous allons manipuler des vecteurs que nous représenterons sous la forme de tableau d'entiers. Voici les tests que vous devez vérifier (il est conseillé d'écrire les fonctions dans le même ordre que l'apparition des tests) :

```
class TP5_Exercice3 extends Program {
  void testReadVector() {
    // En supposant que l'utilisateur saisit les nombres 1, 2 ,3 ...
    assertArrayEquals(new int[]{1,2,3}, saisirVecteur());
  }
  // Ecrire ici la fonction saisirVecteur ...
  void testVectorToString() {
    int[] v1 = new int[]{1,2,3};
    assertEquals("[1,2,3]", vectorToString(v1));
  }
  // Ecrire ici la fonction vecteurToString
  void testAdditionner() {
    int[] v1 = new int[]{1,2,3}, v2 = new int[]{-1,-2,-3};
    assertArrayEquals(new int[]{0,0,0}, add(v1,v2));
  }
  // Ecrire ici la fonction add
  void testScalarProduct() {
    assertEquals(6, scalarProduct(new int[]{1,1,1},
      new int[]{1,2,3}));
    assertEquals(-14, scalarProduct(new int[]{1,2,3},
      new int[]{-1,-2,-3}));
  }
  // Ecrire ici la fonction scalarProduct
  void testEquals() {
    assertTrue(equals(new int[]{1,1,1}, new int[]{1,1,1}));
    assertFalse(equals(new int[]{1,1,2}, new int[]{1,1,1}));
  }
  // Ecrire ici la fonction equals (en ne faisant pas de calculs inutiles !
}
```

Chapitre 3 : Un algorithme plus complexe

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 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 si 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 ?
3. Quand la partie s'arrête-t-elle ? Quelle type de boucle semble le plus approprié ?
4. Concevez l'algorithme permettant de jouer au pendu.