

Programmation Orientée Objets avec Java

Chapitre 1 : les bases de Java

Stéphane Malandain / Yassin Rekik

1.1 Exercice

- Écrire un code qui simule le comportement d'une boucle `while` à l'aide de la boucle `do while`
- Écrire un code qui simule le comportement d'une boucle `do while` à l'aide de la boucle `while`

Utilisez le modèle ci-après :

```
public class Loop {
    public static void whileWithDoWhile(int i) {
        /* essayez par exemple de reproduire l'équivalent suivant:
        while(i < 5) {
            System.out.println(i);
            i++;
        }
        */
    }

    public static void doWhileWithWhile(int i) {
        /* essayez par exemple de reproduire l'équivalent suivant:
        do {
            System.out.println(i);
            i++;
        } while (i < 5);

        */
    }

    Run | Debug
    public static void main(String[] args) {

    }
}
```

1.2 Exercice

Indiquez quelles lignes ne compilent pas :

```
1  short s = 10;  
2  byte b = s;  
3  int i = s;  
4  long l = s;  
5
```

1.3 Exercice

Indiquez quelles lignes ne compilent pas :

```
1  short s = 10;  
2  s = s + s;  
3  s = s * 2;
```

1.4 Exercice

```
1  int i = s++;
```

Que vaut i à la fin de l'exécution ?

1.5 Exercice

On dispose des méthodes suivantes :

```
1  void g(int n, float x) { ... }  
2  void h(short s) { ... }
```

Et des déclarations suivantes :

```
1  int i;  
2  byte b;  
3  float f;  
4  double d;
```

Spécifiez si les appels suivants sont corrects ou non tout en justifiant vos réponses

```

1  g(i, f);
2  g(b+1, f);
3  g(b, f);
4  g(i, d);
5  g(i, i);
6  g(i, 2*f);
7  g(i, 2.0*f);
8  h(b);
9  h(b+1);
10 h(5);
11 h(5.0);

```

Pratique

1.6 Exercice (Triangle)

A l'aide d'un paramètre `n` indiquant la hauteur, écrivez une fonction `printTriangle(int n)` qui affiche un triangle sous cette forme (ici, `n = 4`) :

```

*
**
***
****

```

1.7 Exercice (Sapin)

A l'aide d'un paramètre `n` indiquant la hauteur, écrivez une fonction `printSapin(int n)` qui affiche un sapin sous cette forme (ici, `n = 4`) :

```

  *
 ***
*****
*****

```

1.8 Exercice (Factorielle)

Réalisez une fonction permettant de calculer la factorielle de `n`.

$$n! = \prod_{i=1}^n i$$

1.9 Exercice (Pi)

Réalisez une fonction permettant de calculer une approximation de π :

$$\sum_{n=1}^N \frac{1}{n^4} = \frac{\pi^4}{90}$$

Cette fonction prend n en paramètre et retourne π

1.10 Exercice (Jeu du serpent)

Écrivez un programme qui affiche un serpent. A chaque itération, le serpent est affiché à l'aide d'une suite d'étoiles. Le programme demande ensuite d'augmenter ou de diminuer la taille du serpent. Le programme s'arrête lorsque la taille du serpent est nulle.

Exemple :

```
malandai@StephBook code % java serpent
(^)
Direction:+
*(^^)
Direction:+
**(^)
Direction:+
***(^)
Direction:+
****(^)
Direction:+
*****(^)
Direction:-
****(^)
Direction:-
***(^)
Direction:-
**(^)
Direction:-
*(^^)
Direction:-
(^)
Direction:-
Bye !
```

L'algorithme peut s'énoncer ainsi :

La taille initiale du serpent est de

Tant que la taille est supérieure à 0 faire :

- Afficher le serpent en fonction de sa taille.
- Demander à l'utilisateur d'augmenter ou de diminuer la taille
 - o S'il souhaite augmenter la taille, incrémenter la taille du serpent de 1
 - o S'il souhaite diminuer la taille, décrémenter la taille du serpent de 1

Le programme se termine lorsque la taille est de zéro.

Remarques :

- A vous de définir ce qu'il se passe si l'utilisateur entre une valeur différente de + ou - (quitter le programme, ne rien faire, afficher autre chose qu'un serpent, ...)
- Vous êtes libre d'afficher le serpent sur une ou plusieurs lignes.

- Un + serait d'afficher une tête au serpent (ex. `***0` ou `****x` ou `xxx(^)`)

1.11 Exercice (Vecteurs)

Réalisez un ensemble de fonctionnalités permettant le calcul vectoriel sur des vecteurs. Utilisez un tableau de `doubles` pour représenter un vecteur.

Écrivez les méthodes statiques suivantes :

- La méthode `add`. Elle prend deux vecteurs et retourne leur addition.
Ex. `add(new double[]{1.0, 2.0, 3.0}, new double[]{2.0, 1.0, -5.0})` retourne `double[3]{3.0, 3.0, -3.0}`.
- La méthode `mul` qui multiplie un vecteur par une valeur numérique.
- A l'aide des 2 premières fonctions, écrivez la méthode statique `sub` qui soustrait le deuxième vecteur du premier.
- La méthode `len` qui retourne le nombre de composante d'un vecteur
- `norm` qui retourne la norme (ou la longueur) d'un vecteur.
Par exemple, `norm(new double[]{1.0, 2.0, 2.0})` retourne `3.0`.

$$\vec{v} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \text{ est } \|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

- Le calcul de la norme d'un vecteur
- `sum` qui prend une liste de vecteurs en paramètre et les additionne
- `norms` qui prend une liste de vecteurs en paramètre et retourne la norme de leur addition
- `concat` qui concatène 2 vecteurs
- `sliceFrom` qui retourne un sous-ensemble du vecteur jusqu'à un indice (non compris)
- `slice` qui est la combinaison des deux précédentes (avec un index de début et un de fin)

Remarques :

- Respectez le nommage
- Respectez les conventions de nommage du langage :

<https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>

- Écrivez des méthodes courtes et concises
- Retournez des nouvelles copies de tableaux. Ne modifiez pas vos arguments
- Utilisez les fonctionnalités offertes par la classe `Arrays` :

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html>

- Écrivez vos méthodes statiques dans un fichier `Vector.java` (sans `main`)
- La méthode principale avec vos tests doit se trouver dans un fichier `App.java`
- Les fonctions qui retournent des vecteurs doivent retourner un vecteur vide si les vecteurs entrés en arguments ne sont pas conformes !

1.12 Exercice (Matrices)

Réalisez des fonctionnalités sur les matrices :

- L'affichage d'une matrice
- La multiplication de deux matrices
- L'addition de deux matrices

Retournez une matrice vide en cas d'erreur

1.13 Exercice (Tableaux)

Écrivez une méthode qui compte le nombre d'éléments positifs dans un tableau de tableaux. Un exemple d'utilisation est fourni dans la méthode principale `main` suivante :

```
public class exo1_13 {  
  
    Run | Debug  
    public static void main(String[] args) {  
        double[][] example = {  
            {1.0, -2.0},  
            {3.0},  
            {1.5, -2.5, 3.0}  
        };  
  
        int res = countPositive( example );  
        System.out.println("Count:" + res);  
        // Affiche "Count : 4"  
    }  
}
```

1.14 Exercice (Strings)

Écrivez les fonctionnalités ci-dessous :

```
/* Détermine si un nombre qui se trouve dans une chaîne de caractères contient
 * un entier
 * Note: prend en compte le signe, mais ne prend pas en compte les espaces
 * Exemples:
 * isNumeric("42") -> true
 * isNumeric(" 22 ") -> true
 * isNumeric(" -33 ") -> true
 * isNumeric(" 22.0") -> false
 * isNumeric("2f3") -> false */

public static boolean isNumeric(String term) {
    /* todo */
}

/* Retourne un tableau d'indices où chaque valeur indique la position d'un
 * caractère dans une chaîne
 * Exemples:
 * indexes("maison", 'i') -> {2}
 * isNumeric("tralala", 'a') -> {2,4,6}
 * isNumeric("coucou", 'x') -> {} */

public static int[] indexes(String term, char c) {
    /* todo */
}
```