

Introduction à la Programmation 1

Séance 3 de cours/TD

Université Paris-Diderot

Objectifs:

- Manipuler des boucles avec accumulateurs
- Manipuler deux boucles imbriquées

1 Accumuler des valeurs grâce aux boucles

Utilisation d'accumulateur dans les boucles [COURS]

- Si une variable est déclarée avant une boucle, sa valeur persiste d'une itération de la boucle à la suivante. On peut donc modifier sa valeur à chaque itération de boucle, et donc *accumuler* une valeur dans cette variable.
- Par exemple, le fragment de code suivant calcule la somme des carrés des entiers de 1 à 100 :

```
1 int s = 0;
2 for (int i = 1; i <= 100; i++) {
3     s = s + i * i;
4 }
```

Exercice 1 (Accumulation, ★)

Modifier la boucle donnée ci-dessus pour qu'elle calcule la somme des cubes des entiers de 1 à 42. □

Exercice 2 (Comprendre et modifier une boucle, ★)

```
1 String st = "";
2 for (int i = 1; i <= 50; i++) {
3     st = st + "ab";
4 }
```

- Que contient la variable `st` après la suite d'instructions donnée ci-dessus ?
- Modifier les instructions ci-dessus pour qu'à la fin de leur exécution la variable `st` contienne la chaîne de caractère "aaaaaa ... aaaa" avec la lettre "a" répétée 110 fois.
- Modifier de nouveau la suite d'instructions pour imprimer à l'écran à chaque tour de boucle le contenu la variable `st`. Qu'a cherché votre code ? □

Exercice 3 (Accumulation booléenne, ★★)

On suppose donnée la fonction suivante qui lit un entier au clavier :

```
1 public static int readInt() {
2     Scanner sc = new Scanner (System.in);
```

3
4 }

```
3 |         return sc.nextInt ();  
4 |     }
```

| | |

Écrire un fragment de code qui lit 5 entiers au clavier, puis affiche « Gagné » si l'entier 42 se trouvait parmi ceux-là, et sinon « Perdu ». Attention, il faut lire les 5 entiers et seulement ensuite afficher le résultat. □

2 Boucles imbriquées

Boucles imbriquées _____[9329.90 m 3i.7Tf f 8.126

```
1  /* Affiche un entier. */
2
3  public static void printInt (int x) {
4      System.out.print (x);
5  }
6
7  /* Affiche une chaîne de caractères. */
8
9  public static void printString (String s) {
10     System.out.print (s);
11 }
```

4 Do it yourself

Exercice 6 (Puissance, ★)

Écrire une fonction “`int power (int x, int n)`” qui retourne la valeur x^n .

□

Exercice 7 (Factorielle, ★)

Écrire une fonction “`int fact (int n)`” qui retourne la factorielle de n .

□

Exercice 8 (Ça use, ★★)

1. Écrire un fragment de code qui affiche les paroles de la chanson que vos neveux chantaient durant les vacances d'été :

1 kilomètre à pied, ça use les souliers.

2 kilomètres à pied, ça use les souliers.

...

100 kilomètres à pied, ça use les souliers.

Attention, le premier vers est différent (le mot kilomètre est au singulier).

2. Modifier votre code pour qu'il affiche les vers dans l'ordre inverse.

□

Exercice 9 (Nombres premiers, ★★)

1. Écrire une fonction “`int prime (int n)`” qui retourne 1 si n est un nombre premier, 0 sinon. (On rappelle que 1 est un nombre premier.)
2. Écrire une fonction “`int sum_prime (int n)`” qui retourne la somme des nombres premiers compris (au sens large) entre 1 et n .

□

Exercice 10 (Table de multiplication, ★★)

1. Écrire un fragment de code qui affiche les tables de multiplication pour les entiers de 1 à 10.

1 2 3 4 ... 10

2 4 6 8 ... 20

...

10 20 30 ... 100

2. Écrire une fonction qui prend un entier n en paramètre et retourne 1 si n apparaît dans la table de multiplication, et 0 sinon.

□

Exercice 11 (Carrés, **)

1. Écrire une fonction qui prend en entrée un entier positif n et affiche un carré plein de côté n . Par exemple, si n vaut 4, votre fonction affichera

```
****
****
****
****
```

2. Modifier votre fonction pour qu'elle affiche un carré creux, par exemple pour n valant 4,

```
****
*  *
*  *
*  *
****
```

□

Exercice 12 (Séries, * – ***)

Pour chacune des séries suivantes, trouver le plus petit programme (en nombre d'appels de fonctions) qui affiche les séries de 20 nombres suivants à l'écran :

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (*)
- 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 (*)
- 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 (***)
- 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 (**)
- 0 2 4 6 8 9 11 13 15 17 18 20 22 24 26 27 29 31 33 35 (***)
- 0 4 2 2 1 2 3 3 4 12 5 17 9 2 1 3 4 19 3 8 (*)

□