

# TP 1\_2 : Éléments de base en Java

Objectifs : Dans ce TP, nous verrons comment lire et écrire des informations à la console et passerons en revue quelques structures de contrôle.

## Exercice 1 : Hello étudiant !

Cet exercice, similaire à l'exercice 1 du TP1\_1, vous enseignera comment lire une information à la console.

1. Créez un nouveau projet si nécessaire, ou reprenez le projet « Cours Java » du TP1\_1 (recommandé). Créez un nouveau package nommé « tp1\_2 » puis créez une classe nommée Exercice1, possédant un point d'entrée au programme (méthode main publique et statique).
2. Dans la méthode main, ajoutez le code permettant d'écrire « Exercice 2 » et à la ligne « Entrez votre nom : ». Testez.
3. Toujours dans la méthode main, tapez le code suivant :

```
Scanner lecteur = new Scanner(System.in);
```

« Scanner » devrait être souligné en rouge, car il n'est pas reconnu par le compilateur. Lorsque vous passez le curseur sur le mot, le message suivant devrait apparaître :



4. Cliquez sur « Import 'Scanner' (java.util) » pour résoudre le problème. Vous remarquerez que Java ajouté seul une nouvelle ligne en haut du fichier :

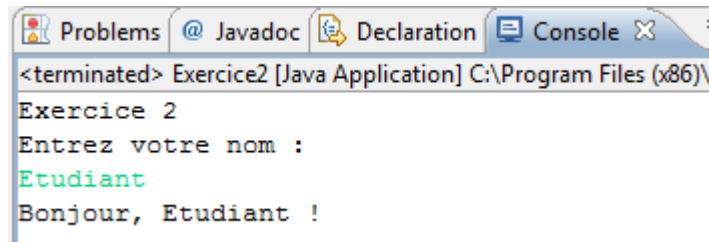
```
import java.util.Scanner;
```

Cela signifie que lorsque « Scanner » est utilisé, le terme fait référence à « java.util.scanner ». De la même façon, « System.out » pourrait s'écrire « java.lang.System.out ». Toutes les classes utilisées doivent être importées, sauf celles situées dans le paquetage courant (ici « tp1\_2 ») ou dans « java.lang ». Pour faire appel à la classe « Point » du TP1\_1, il faudrait importer « tp1\_1.Point ».

5. Vous avez instancié en question 4 une classe servant à lire des informations depuis la ligne de commande. Vous pouvez à présent lire le contenu d'une ligne en écrivant :

```
lecteur.nextLine()
```

Ecrivez le code permettant de lire votre nom à la ligne de commande et afficher ensuite « Bonjour, <votre nom> ! ». Voici un exemple de résultat attendu :



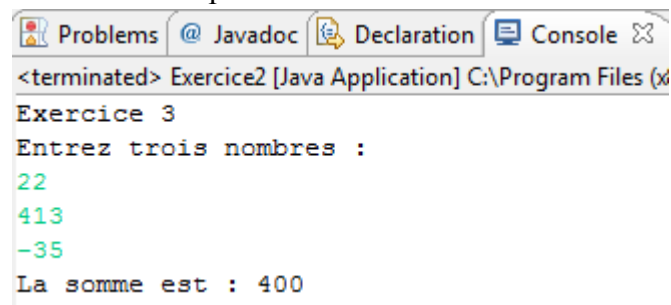
```
<terminated> Exercise2 [Java Application] C:\Program Files (x86)\
Exercise 2
Entrez votre nom :
Etudiant
Bonjour, Etudiant !
```

Pour entrer votre nom, il se peut que vous ayez à cliquer à l'intérieur de la fenêtre de la console. Vous terminerez votre saisie par la touche retour à la ligne.

## Exercice 2 : Additions

Dans cet exercice, vous apprendrez à convertir des chaînes de caractères en entiers et utiliserez les structure de contrôle *if*, *for* et *while*.

1. A partir de l'exemple de l'exercice 1, lisez trois nombres depuis la console et affichez leur somme. Pour convertir une chaîne de caractères en entier, utilisez la méthode *Integer.parseInt* dont vous trouverez les paramètres et le type de retour dans la Javadoc. Sachez qu'il est possible, de façon similaire, de convertir une chaîne en Double, Float etc. Voici un exemple de résultat :



```
<terminated> Exercise2 [Java Application] C:\Program Files (x86)\
Exercise 3
Entrez trois nombres :
22
413
-35
La somme est : 400
```

2. Modifiez votre programme pour qu'il commence par demander combien de nombres doivent être saisis, lise autant de nombre que demandé, puis affiche leur somme. Vous essaieriez deux méthodes différentes : en utilisant une boucle *for*, puis en utilisant une boucle *while*.
3. Modifiez à nouveau votre programme pour que la quantité de nombres saisis soit comprise strictement entre 1 et 10.
4. Modifiez votre programme pour qu'il n'y ait plus besoin de demander combien de nombres doivent être saisis : il doit à présent s'arrêter lorsque l'utilisateur entre une ligne vide. Pour tester si une chaîne de caractères est vide, vous pouvez utiliser la méthode *String.isEmpty*.  
Conseil : la logique est la suivante : « tant que la chaîne entrée n'est pas vide, ... »
5. Que se passe-t-il lorsque l'on entre du texte au lieu d'un nombre ? Analysez et décrivez les différents éléments du message d'erreur produit. Nous verrons dans un prochain TP comment traiter automatiquement cette erreur.

### Exercice 3 : Break et continue

Dans cet exercice, vous expérimenterez les différences entre les instructions *break* et *continue*.

1. Que produit le programme suivant à la console ? Tentez d'abord de répondre à la question, puis recopiez le programme et exécutez-le. Comparez le résultat obtenu avec celui attendu. Formulez une explication sur le fonctionnement des instructions *break* et *continue*.

```
public class Exercice3 {  
    public static void main(String[] args) {  
        int n = 0;  
        while(n < 15) {  
            n++;  
            if(n % 2 == 0) {  
                System.out.println(n + " est pair");  
                continue;  
            }  
            if(n % 3 == 0) {  
                System.out.println(n + " est multiple de 3");  
            }  
            if(n % 7 == 0) {  
                System.out.println(n + " est multiple de 7");  
                break;  
            }  
        }  
    }  
}
```

### Exercice 4 : Récursivité avec Fibonacci (facultatif)


Cet exercice vous permettra de tester les appels récursifs en Java.

La suite de Fibonacci se définit mathématiquement comme suit :

$$fib(0) \Rightarrow 0, fib(1) \Rightarrow 1, fib(n) = fib(n-1) + fib(n-2)$$

1. Créez une classe nommée Fibonacci. Ecrivez la méthode statique *fib*, qui prenne en paramètre un nombre entier n et qui renvoie 0 pour  $n \leq 0$  et la valeur calculée de *fib*(n) pour  $n \geq 1$ . Cette méthode est dite récursive.
2. Créez une méthode main qui calcule toutes les valeurs de *fib*(n) pour n allant de 0 à 100 et les affiche à la console. On attend que le résultat à la console soit de la forme :

```
fib(0) = 0  
fib(1) = 1  
fib(2) = 1  
fib(3) = 2  
fib(4) = 3  
fib(5) = 5
```

3. A partir de quelle valeur de n *fib*(n) met-il plus d'une seconde à s'afficher ? Vous pouvez à tout moment stopper l'exécution en appuyant sur le bouton .
4. Pour comprendre pourquoi le programme devient rapidement lent, nous allons chercher à compter le nombre d'appels à *fib* pour une valeur de n donnée. Sachant que pour n=0 ou 1, *fib*(n) n'exécute pas d'appel récursif ; que pour n=2, *fib*(n) effectue

deux appels ( $fib(n-1)$  et  $fib(n-2)$ ), on peut en déduire la suite *appels* qui compte le nombre d'appels à *fib* et qui a la forme suivante :

$appels(0) \Rightarrow 1$  (on appelle une seule fois *fib*)

$appels(1) \Rightarrow 1$

$appels(2) \Rightarrow 1 + 2$  (*fib* exécute deux sous-appels)

$appels(n) \Rightarrow 1 + appels(n-1) + appels(n-2)$

Créez la méthode *appels* correspondant à cette définition. Combien d'appels sont réalisés pour  $fib(20)$  ? Pour  $fib(30)$  ? Comment qualifiez-vous la complexité de ce type d'algorithme ?

5. On souhaite maintenant vérifier si *appels* donne un résultat juste. Pour cela, nous allons créer un compteur statique dans la classe hébergeant la méthode *fib*. A chaque appel de la méthode *fib*, celle-ci doit incrémenter le compteur. Entre deux appels de *fib*, il est nécessaire de remettre à zéro le compteur.

Modifiez votre méthode *main* pour, à chaque itération de  $n$ , calculer  $fib(n)$  et comparer les valeurs de *compteur* et *appels(n)*. Pour  $n=3$ , le programme doit afficher :

|                                   |
|-----------------------------------|
| $fib(2) = 3$ appels, compteur = 3 |
|-----------------------------------|

Vérifiez que le compteur et *appels(n)* donnent bien le même résultat quel que soit la valeur de  $n$  (calculable en un temps raisonnable).