

# TP LES ITÉRATIONS.

Dans ce TP nous allons utiliser une famille d'algorithme dit de CUMUL très utilisé lorsqu'on utilise des boucles. Son utilisation est souvent très intuitive comme dans le cas d'une variable « i » utilisée comme compteur dans une boucle :

```
int i = 0 ;
while (i<10){
    System.out.println("N ?");
    i = i + 1;    //➔ Ici on réalise un CUMUL.
}
```

Détaillons la ligne mise en commentaire. La variable « i » est utilisée pour réaliser un calcul « i+1 », et dans le même temps, « i » est utilisé pour recevoir le résultat de ce calcul « i = ... ». C'est une opération de cumul. On ajoute 1 à « i » à chaque itération.

Dans les exercices suivant le 1,2 et 3 sont des problèmes de type « **CUMUL** », le 4 et 5 de type « **DECOMPOSITION** ».

De plus l'utilisation des boucles en programmation est souvent source de « bug » qu'il faut corriger. Ce sera ici l'occasion de mettre en œuvre l'utilisation d'un outil dédié à la recherche de bug : « le **DEBUGER** ».

## EXERCICE 1

Écrivez un programme qui lit N nombres entiers au clavier et qui affiche leur somme. Le nombre N est choisi par l'utilisateur dans une zone texte. Ensuite, chaque valeur est saisie à l'aide d'un prompt.

Saisir le code suivant correspondant à l'exercice :

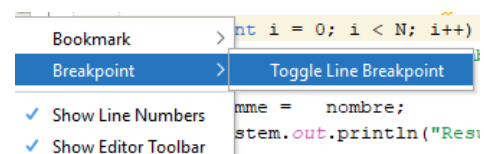
```
public static void ex01() {
    Scanner sc = new Scanner(System.in);
    System.out.println("N ?");
    int N = sc.nextInt();

    int somme = 0, nombre = 0;
    for (int i = 0; i < N; i++) {
        System.out.println("nombre " + i + " ? ");
        nombre = sc.nextInt();
        somme = nombre;
        System.out.println("Resultat intermédiaire : " + somme);
    }
    System.out.println("\nResultat Final : " + somme);
}
```

Comparer le résultat de ce programme avec la vidéo du résultat attendu : [vidéo](#)

Conclusion : le programme fourni ne fait pas ce qui est attendu. Nous allons le débbugger en suivant les étapes suivantes pour trouver notre erreur.

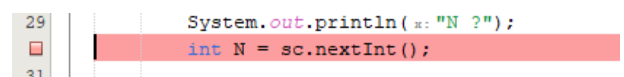
- **Étape 1 : placer un point d'arrêt.** Procéder comme illustré. Clic droit dans la zone des numéros de lignes, puis Breakpoint. Le point d'arrêt sera placé sur la ligne « int N=sc.nextInt(); » .





Une autre façon de placer un point d'arrêt est d'effectuer un simple clic sur le numéro de ligne.

Une dernière solution est la touche de raccourci clavier CTRL+F8.

Une fois placé, le point d'arrêt est affiché avec un carré rose, et la ligne surlignée en rose également. Le placement du point d'arrêt n'est pas nécessairement la 1<sup>re</sup> ligne de votre code, mais sur celle qui semble être à l'origine du problème.







- **Étape 2 : lancer le programme**  en mode DEBUG (CTRL + F5) . L'exécution du programme s'arrête sur votre point d'arrêt. Votre programme est comme en "pause" et attend une action de votre part. On note visuellement le curseur  indiquant où se situe le programme dans son exécution. La ligne est maintenant surlignée en vert comme illustré.

```

29      System.out.println( "N ?" );
30      int N = sc.nextInt();
31
      int somme = 0, nombre = 0;

```

- **Étape 3:** faire du pas-à-pas. Vous disposez de 4 actions :

-  **Continue** (F5), permet de lancer le programme jusqu'au prochain point d'arrêt. Si aucun point d'arrêt n'est trouvé, le programme s'exécute normalement.
-  **StepOver** (F8), permet d'exécuter la ligne en cours.
-  **StepInto** (F7), permet d'exécuter l'intérieur de la ligne en cours. A le même effet que **StepOver** si vous êtes sur une ligne ne comportant pas de fonctions écrites par le programmeur.
-  **StepOut** (CTRL+F7), permet de sortir d'une fonction et de revenir au programme principal. Même effet que **Continue** si la ligne en cours ne correspond pas à une fonction.

Utiliser ses 4 actions pour comprendre le fonctionnement du debugger

- **Étape 4 :** 2 fenêtres sont très utiles en phase de débogage. Il s'agit de la fenêtre "Variables" et "Watches".

Elles permettent d'observer le contenu des variables lors de l'exécution du programme et ainsi de repérer tout comportement suspect. Ici, nous voyons nos variables. Par exemple, N est un « int » et sa valeur est « 5 ».

Variables x			
Watches Breakpoints Evaluation Result			
	Name	Type	Value
Static			
>	class	Class	class javaapplication2.JavaApp...
>	sc	Scanner	#212
	N	int	5
	somme	int	0
	nombre	int	0
	i	int	0

Maintenant que vous avez compris le fonctionnement du DEBUGGER corriger les 2 erreurs qui se sont glissées dans notre code.

## EXERCICE 2

Reprendre l'exercice précédent. Cette fois-ci on ne demande pas à l'utilisateur le nombre (N) de sommes à effectuer. Lorsqu'il saisit la valeur -1 le programme s'arrête et affiche le résultat. Cela suppose que l'utilisateur ne peut saisir que des valeurs > 0

Exemple de trace d'exécution :

```

Saisir une suite de nombre. Saisir -1 pour quitter.
Nombre 1: 10
Nombre 2: 12
Nombre 3: 1
Nombre 4: -1
La somme des 3 nombres est: 23

```

## EXERCICE 3

Écrire un programme qui calcule N! (Factorielle de N). Le calcul se fait selon l'opération suivante :  $1 \times 2 \times 3 \times \dots \times N \times (N-1)$

Cas particulier 0!=1

Exemple avec N=5 →  $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

## EXERCICE 4

Nous avons déjà écrit un programme permettant de convertir un nombre binaire (de 4 chiffres maximum) en décimal.

Cette fois-ci aucune limite du nombre de chiffres n'est fixée, pour cela on choisira un type « long ».

Exemple : 1001011011(2) → 603 (10)

Ce problème utilise les boucles c'est évident, mais reste un problème classique de décomposition. Pour le résoudre, on procèdera par division successive avec le nombre 10 et l'on récupère ainsi chaque digit.

## EXERCICE 5

Écrire un programme permettant de convertir un nombre en base 10 vers la base 2. Là encore un problème de décompositions avec division successive par 2.

Exemple : 603 (10) → 1001011011(2)

## EXERCICE 6

Écrire un programme permettant d'afficher un triangle isocèle formé d'étoiles de N lignes (N est fourni au clavier).

Exemple avec N=8

*	→ ici 7 espaces + 1*
***	→ ici 6 espaces + 2*
*****	→ ici 5 espaces + 5*
*****	→ ici 4 espaces + 7* et ainsi de suite
*****	
*****	
*****	
*****	