

# Chapitre 13

La récursivité

1

Définition et principes

2

Exemples d'applications

1

Définition et principes

2

Exemples d'applications

## Définition et principes (1/5)

- La récursivité n'est pas un concept Java en soi, mais un paradigme algorithmique transverse à beaucoup de langages de programmation impératifs et qui tire ses sources dans le principe du **raisonnement par récurrence en mathématique**
- Pour rappel, en mathématique, un phénomène dit récurrent est un phénomène **répétitif dont l'état précédent induit l'obtention de l'état suivant**

### Exemple :

Considérons la déclaration suivante : « *S'il fait beau toute une journée alors il fera beau toute la journée du lendemain* »

> Cette déclaration peut être considérée comme un raisonnement par récurrence, car si on pose la propriété

$P_n$  = « **Il fait beau toute la journée  $n$**  », alors en se basant sur cette déclaration, on peut dire que comme  $P_n$  est vraie, alors

$P_{n+1}$  = « **Il fait beau toute la journée  $n+1$**  » est aussi vraie :  $\rightarrow (P_n \text{ est vraie}) \Rightarrow (P_{n+1} \text{ est vraie})$

- **Définition du principe de récurrence :**

Soit une propriété  $P_n$

Si  $(P_n \text{ est vraie})$  alors  $(P_{n+1} \text{ est vraie})$  et s'il existe un entier  $q$  tel que  $P_q$  est vraie, alors pour tout entier  $n \geq q$ ,  $P_n$  est vraie.

## Définition et principes (2/5)

- Exemple de raisonnement par récurrence sur un problème mathématique :

Considérons la problématique de la somme des entiers consécutifs :  $\sum_{k=0}^n = 0 + 1 + 2 + \dots + n$

Comment démontrer que la propriété  $\sum_{k=0}^n = \frac{n(n+1)}{2}$  est vraie pour tout entier positif  $n$  quelque soit sa valeur ?

> Le raisonnement par récurrence pour démontrer cela se fait comme suit :

- supposons  $n = 0$ , alors  $\sum_{k=0}^0 = 0$  et  $\frac{0(0+1)}{2} = 0 \Rightarrow 0 = 0$ , donc la propriété est vraie pour  $n = 0$

- Supposons que la propriété est vraie pour  $q = n-1$  :  $\sum_{k=0}^{n-1} = \frac{(n-1)((n-1)+1)}{2} = \frac{(n-1)n}{2}$  et montrons qu'elle est vraie pour  $q = n$

- Pour  $q = n$ ,  $\sum_{k=0}^n = \sum_{k=0}^{n-1} + n = \frac{(n-1)n}{2} + n = \frac{(n-1)n}{2} + \frac{2n}{2} = \frac{(n-1+2)n}{2} = \frac{(n+1)n}{2} = \frac{n(n+1)}{2}$

CQFD !

## Définition et principes (3/5)

- Mais comment tout ça se traduit-il en programmation ?

>>> C'est très simple : il suffit de voir une propriété  $P_n$  comme une fonction (une méthode Java donc)  $P$  ayant un paramètre  $n$ .  
Et comme l'état de  $P_n$  induit l'état de  $P_{n+1}$ , alors l'assertion  $(P_n \text{ est vraie}) \Rightarrow (P_{n+1} \text{ est vraie})$  se converti en :

$$P_{n+1} = P(P_n) \quad \left\{ \begin{array}{l} P_0 = x \\ P_1 = P(P_0) \\ P_2 = P(P_1) \\ \dots \\ P_{n+1} = P(P_n) \end{array} \right.$$

>> En d'autres termes, la récursivité en programmation est une technique qui permet à une méthode de s'auto-appeler répétitivement avec pour but de réutiliser le résultat de l'appel précédent dans l'appel suivant

```
public int P(int n+1) {  
    .  
    .  
    x = P(n);  
    y = faireQuelquechoseAvec(x);  
    return y;  
}
```

• Appel de la méthode  $P(n)$  dans la méthode  $P(n+1)$

$P(n+1)$  est donc fonction/dépendant de  $P(n)$ , ce qui peut aussi se traduire par la représentation  $P(n+1) = f(P(n))$  où  $f$  correspond à l'ensemble des traitements effectués dans le corps de la méthode  $P$  en tenant compte du résultat de  $P(n)$

## Définition et principes (4/5)

- Mais attendez, puisque la méthode P s'auto-appelle répétitivement et indéfiniment, comment arrêter ce cercle vicieux ?

- > En effet, il y a un risque lorsqu'on conçoit une méthode avec un fonctionnement récursif. Si l'on ne réfléchit pas et n'implémente pas comment ces appels imbriqués vont finir par s'arrêter et rendre un résultat final, alors ils ne se termineront jamais et la conséquence est que le programme va se figer et planter pour saturation de la mémoire. Pour information, le résultat d'un appel de méthode en Java est stocké dans la pile mémoire de la JVM.

- > Pour résoudre ce problème, on préconise au développeur l'utilisation d'une **condition de terminaison**

- >> La condition de terminaison implique des **contraintes à respecter** lorsqu'on définit une méthode récursive :

- S'assurer qu'il existe une relation d'ordre (montante ou descendante) entre les paramètres  $n$  et  $n+1$


- => c'est la notion de monotonie croissante ou décroissante en mathématique

- => cette relation d'ordre va donc assurer que la méthode s'auto-appelle avec des paramètres dont on maîtrise au moins le sens de leurs évolutions (croissante ou décroissante)

- Connaissant donc l'ordre, on peut prévoir une condition d'arrêt de la récursion dans le corps de la méthode P

- On utilise les structures conditionnelles **if..else** pour implémenter cette condition d'arrêt

## Définition et principes (5/5)

- Mais attendez, pourquoi s'emberlificoter avec la récursivité, il existe les structures de boucle, non !!!? 
- > En effet, on a dit que la récursivité fait de la *répétition de traitement*, mais impose en plus des conditions et des risques. Pourquoi ne pas faire simplement avec les boucles ?
- > Malheureusement, la réponse n'est pas tranchée. Les structures de boucles ont aussi leurs limites et ne peuvent répondre à toutes les problématiques nécessitant un traitement répétitif.
  - La boucle **for** impose de connaître au préalable le nombre d'itérations à effectuer. Ce qui n'est pas le cas avec la récursivité
  - Avec les boucles **for**, **do..while** et **while**, chaque itération est indépendante de l'autre. Il est donc souvent difficile de réexploiter le corps de la boucle d'une itération comme paramètre de l'itération suivante, ce qui est plutôt très simple avec la récursivité.
  - Mais la récursivité a aussi ses inconvénients qui sont d'ordre technique et sous-jacente à la JVM, car son utilisation abusive et mal-conçue peut engendrer des problèmes de performance sur le programme.
- > Certaines opérations effectuées avec les boucles peuvent être converties en traitements récursifs et vice-versa. Le choix et la conception dépendent exclusivement de la vision et des objectifs du développeur. Il n'y a pas de meilleur choix à priori entre les deux approches



1

Définition et principes

2

Exemples d'applications

# Problème

Considérons la formule du factoriel suivante : .

$$\begin{cases} 0! = 1 \\ n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1 \end{cases}$$

Ecrire un programme Java qui calcule le factoriel de tout nombre positif n saisi par l'utilisateur sur le clavier.

# Solution

```
import java.util.Scanner;

public class Factoriel {

    //méthode récursive
    public long facto(long n) {

        //condition de terminaison
        if (n == 0) {
            return 1;
        } else {
            //appel récursif facto(n - 1)
            return n * facto(n - 1);
        }
    }

    public static void main(String[] args) {
        Factoriel fact = new Factoriel();

        System.out.println("Entrez un nombre positif n : ");
        Scanner scan = null;
        try {
            scan = new Scanner(System.in);
            if (scan.hasNextLong()) {
                long n = scan.nextInt();
                if (n < 0) {
                    throw new Exception("Le nombre doit être positif");
                }
                long resultat = fact.facto(n);
                System.out.println("Factoriel de n=" + n + " : " + resultat);
            }
        } catch (Exception e) {
            System.out.println("Erreur de saisie " + e.getMessage());
        } finally {
            scan.close();
        }
    }
}
```

La saisie de n = 10, retourne 39916800 comme résultat

# Problème

Considérons la suite numérique suivante : .

$$\begin{cases} u_0 = 2 \\ u_n = 3u_{n-1} - 5 \end{cases}$$

Ecrire un programme Java qui implémente cette suite numérique en utilisant la récursivité et qui dans la méthode main demande à l'utilisateur de saisir un nombre entier positif n et calcule  $u_n$ .

# Solution

```
import java.util.Scanner;

public class SuiteNumerique {

    //méthode statique récursive
    public static int suiteU(int n) {

        //condition de terminaison
        if (n == 0) {
            return 2;
        }

        //appel récursif suiteU(n-1)
        return 3 * suiteU(n - 1) - 5;
    }

    public static void main(String[] args) {

        System.out.println("Entrez un nombre positif n : ");
        Scanner scan = null;
        try {
            scan = new Scanner(System.in);
            if (scan.hasNextInt()) {
                int n = scan.nextInt();
                if (n < 0) {
                    throw new Exception("Le nombre doit être positif");
                }
                int resultat = SuiteNumerique.suiteU(n);
                System.out.println("Résultat Un avec n=" + n + " : " + resultat);
            }
        } catch (Exception e) {
            System.out.println("Erreur de saisie " + e.getMessage());
        } finally {
            scan.close();
        }
    }
}
```

La saisie de n = 75, retourne -464945531 comme résultat

