

# Cours 3

Lire la page division entière (voir cours 2.2 ci-dessus)

## I. Les boucles

### I.1 La boucle TANT QUE (*fondamentale*)

I.1.1 En pseudo-langage : TANT QUE condition FAIRE instructions FIN TANT QUE  
répéter les instructions tant que la condition est vraie c'est-à-dire jusqu'à ce que la condition devienne fausse

I.1.2 Exemple de la factorielle : SAISIR(n)  $f \leftarrow n$  TANT QUE  $n > 2$  FAIRE  $n \leftarrow n-1$   $f \leftarrow f \times n$   
FIN TANT QUE AFFICHE(f)

I.1.3 Caractéristiques :

- nombre de tours inconnu a priori
- test au début  $\Rightarrow$  peut tourner 0 fois
- pb d'initialisation de l'expression booléenne
- pb de modification de l'expression booléenne
- comme pour le `if`, toujours mettre des `{ }`

I.1.4 Syntaxe : `while ( expression_booléenne ) { instructions_tant_que_vraie }`  
syntaxe ressemble au `if` mais test  $\neq$  répétition

## II. Nouveau type primitif : nombre "réel"

II.1 type entier = sous-ensemble de  $\mathbf{Z}$  :  $[-2^{31}, +2^{31}-1]$

II.2 type "réel" = sous-ensemble de  $\mathbf{D}$  et non  $\mathbf{R} \Rightarrow$  précision limitée  $\Rightarrow$  nombres à virgule flottante.  
min / max et nombre maximum de chiffres significatifs

exemple : bien que  $\max=10^{22}$ , 1234567 n'est pas représentable si 6 ch.sig.  
ligne réelle (voir dessin)

II.3 Java : type primitif sur 8 octets : `double`

II.4 Représentation binaire :

1|11|52 bits  $\rightarrow$  signe | exposant ( $< 10^{\pm 308}$ ) | mantisse (16 ch.sig.)

beaucoup plus complexe : voir [représentation des nombres réels](#) et pour encore plus d'explications lire [floating point numbers](#)

somme d'inverses de puissances de deux différentes  $\Rightarrow$  5.1 non exact et calculs non exacts.

II.5 Valeurs littérales :

première forme =  $\pm$ chiffres. |  $\pm$ .chiffres |  $\pm$ chiffres.chiffres

deuxième forme =  $\pm$ entierE $\pm$ entier | *première forme*E $\pm$ entier

exemples : 0.5 -4. .32 1E-6 2.5E12

II.6 Opérateurs

II.6.1 unaires : + -

II.6.2 binaires : + - \* / %

/ = division réelle  $\Leftrightarrow$  au moins un des deux nombres est réel

% = définition du modulo étendue aux réels : toujours la relation  $a = b \times q + r$   
mais avec q toujours entier

II.7 Fonctions mathématiques (dans la classe Math)

`sqrt` = square root = racine carrée

`abs`, `sin`, `cos`, `tan`, `log`, `log10`, `exp`, `pow`, ... voir syntaxe d'appel ci-dessous (III.1)

II.8 Comparaisons

- < et > : OK

- <=, >=, ==, != : INTERDIT ! (*problèmes de précision*)

-  $|x-y| < \epsilon$  remplace `x==y`

- encore mieux en tenant compte de l'ordre de grandeur :  $|(x-y)/y| < \epsilon$  ou  $|x/y - 1| < \epsilon$

## II.9 Conversions

- avec perte : `int vPE = (int)vR;` (*nouvelle expression entière, ce n'est pas la partie entière, c'est une troncature*)  
Exemples : `(int)3.14 → 3` `(int)-3.14 → -3` `Partie_entière(-3.14) → -4`
- sans perte (*car  $31 < 52$* ) : `vR = (double)vE;` (*pour la division, nouvelle expression réelle*)
- en `String` : `vS = new String(vR);` **OU** `vS = "" + vR;`
- possible `String → double` **OU** `int`

## III. Membres de classe (≠ membres d'instance)

### III.1 Méthodes de classe

- ne nécessitent pas d'instance pour être appelées ⇒ sont appelées directement sur la classe
- Exemple : `vR = Math.sqrt(vX);`
- ne peuvent pas accéder aux attributs d'instance !
- ⇒ ne peuvent pas appeler de méthodes d'instance
- (par contre une méthode non statique peut appeler une méthode statique)

### III.2 Syntaxe :

- On les reconnaît dans la javadoc car signature commence par `static`
- On les déclare en ajoutant `static` après `public`

### III.3 Attributs de classe

- On désire compter le nombre de cercles.
- Si attribut d'instance `aNbCercles`, valeur dupliquée dans chaque cercle, et si le constructeur l'incrémente, vaut 1 dans tous les cercles !
- On a besoin d'un attribut partagé par tous les objets de la classe `Cercle`.

### III.4 Syntaxe :

- On les reconnaît dans la javadoc car leur déclaration commence par `static`
- On les déclare en ajoutant `static` après `private`

### III.5 Bloc :

- Mais comment les initialiser avant même qu'un objet soit créé ?
- `static { instructions }` à mettre en tête de classe, un peu l'équivalent pour la classe du constructeur pour l'objet : exécuté automatiquement une seule fois au chargement de la classe

### III.6 Applications : les constantes

- Déjà vu : `private final int MAX = 10;`
- mais même valeur dupliquée dans toutes les classes
- ⇒ constante de classe : `public static final int MAX = 10;` (*aucun risque*)

## IV. Paquetages : [A lire](#)

## V. Tests unitaires avec JUnit : [A lire](#)

### Lire le poly :

tout jusqu'à la section 2.2, sections 3.1, 4, 5.2.0, 6, 7.1, 7.2, 8.1, 8.2.1.1, et annexes 6 & 7 (à compléter)

