JAVA - Révisions

1 La base

1.1 Les variables

Kesako ? Les variables stockent des informations d'un type bien précis. Elles se déclarent au début, de la façon suivante (ici on déclare 2 variables du même type) :

type nomVariable1, nomVariable2 ;

type Le type des variables dépend de ce qu'on met dedans :

- **int** = entier
- **double** = réel
- **char** = caractère
- **boolean** = booléens (true/false)

Initialisation Pour initialiser une variable lors de sa création (une pierre deux coups), il suffit de lui affecter une valeur dans la foulée :

int nomVariable1 = 2 ;

1.2 Les conditions

Kesako ? Parfois, on ne va réaliser une opération **que si** on remplit une certaine condition.
C'est ce qu'on traduit en programmation avec **if** (c'est de l'anglais) :

```
if(condition) {          //Si jamais la condition est vérifiée...
  operations;            //... on réalise ces opérations
} else {                 //Sinon (dans tous les autres cas)...
  operations;            //... on réalise ces opérations
}
```

condition Il s'agit de la condition à vérifier, il peut s'agir :
- d'une **égalité** ⇒ `if(varChar == 'a')`
- d'une **différence** ⇒ `if(varChar != 'd')`
- d'une **inégalité** ⇒ `if(varInt >= 3)`
- d'une **combinaison des deux**
  - condition1 **OU** condition2 ⇒ `if(varInt < 2 || varInt > 9)`
  - condition1 **ET** condition2 ⇒ `if(varChar != 'd' && varInt > 9)`

A savoir On peut mettre autant de **if** que de conditions.
Le **else** n'est pas indispensable. Il permet cependant de traiter tous les autres cas.
On peut également utiliser **switch** dans certains cas... mais **if** fonctionne très bien et suffit pour valider haut la main.

1.3 Les boucles for

Kesako ? Parfois on a besoin de **répéter n fois** une opération.
Pour cela on utilise **for** :

```
for(int i=0; condition; evolution) {        //Réalise une boucle
  operations;                               //... sur ces opérations
}
```

condition Ici, on a initialisé un compteur entier i qui débute à 0 (ce sera toujours le cas pour nous). Il faut donc indiquer une condition pour terminer la boucle. **Tant que cette condition est réalisé**, la boucle reprend.

evolution Il s'agit de dire comment évolue le compteur à la fin d'une boucle. Généralement il s'agit d'incrémenter le compteur de 1, on utilise alors **i++**

1.4 Les boucles while

Kesako ? Parfois, on a besoin de répéter une opération **tant qu'une** condition est remplie.
C'est ce qu'on traduit en programmation avec **while** (ça aussi c'est de l'anglais) :

```
while(condition) {        //Tant que la condition est satisfaite
  operations;             //... on réalise ces opérations
}
```

condition Cf. *Les conditions* (même combat).

Do... while Une variante bien utile permet de réaliser la série d'opérations **au moins une fois** (puis **tant que** la condition est remplie).
En pratique, ça sert pour afficher quelque chose et le ré-afficher tant que la réponse de l'utilisateur est différente de celle souhaitée :

```
do {
  operations;
} while(condition);     //On n'oublie pas le ; à la fin de cette ligne !
```

1.5 Ecrire/Lire

Ecrire Pour afficher quelque chose à l'écran, la syntaxe est la suivante :

System.out.println("Texte affiché "+*variable*+" encore du texte !") ;

A savoir Cette syntaxe met par défaut un retour à la ligne à la fin, c'est (très souvent) plus pratique.
Cette syntaxe présente la façon d'afficher du texte ET le contenu d'une variable à l'écran. En résumé, il ne faut pas de guillemets autour de la variable à afficher, et il faut mettre un + entre le texte brut et le nom de la variable.

Lire Pour lire une donnée saisie par l'utilisateur, il faut lui demander (en écrivant la question), puis utiliser cette syntaxe pour affecter la donnée à la bonne variable du programme :

nomVariable = Lire.**type**() ;

Lire.type() Il faut remplacer **type** par la lettre correspondant au type de variable qu'on enregistre :

- **Lire.i()** pour **int**
- **Lire.d()** pour **double**
- **Lire.c()** pour **char**
- **Lire.b()** pour **boolean**

A savoir La donnée enregistrée doit être du même type que la variable que l'on affecte (logique).
Cette version édulcorée ne fonctionne qu'à l'INT, avec la **class** Lire qui va bien (en pratique c'est pas tout à fait ça).

2 Les tableaux

2.1 Créer un tableau

Intérêt Les tableaux permettent en fait de **stocker plusieurs variables** qui sont "de même nature".
Par exemple, un tableau du solde d'un compte bancaire servira à enregistrer les valeurs que prendra ce solde (et de pouvoir faire un historique).

On déclare un tableau de la manière suivante :

type [ ] nomTableau = new **type**[**taille**] ;

type Le type des éléments du tableau :

- **int** = entier
- **double** = réel
- **char** = caractère

taille La taille du tableau (nombre d'éléments inside).

2.2 Ecrire dans un tableau

Il suffit d'affecter une valeur à un élément du tableau (qui n'est qu'une variable) :

nomTableau[**indice**] = **valeur** ;

indice Commence à **0** pour le premier élément du tableau.
Pour un tableau de taille **n**, l'indice du dernier élément est **n-1**.

Attention Il faut prendre garde à ce que la valeur soit du bon type.

2.3 Lire un tableau

Il s'agit simplement de pointer un élément du tableau, afin d'en récupérer la valeur :

System.out.println("La valeur du tableau est "+nomTableau[**indice**]) ;

2.4 Astuce

Boucler un tableau Dans le CF vous aurez besoin de réaliser une boucle afin de **pointer tous les éléments du tableau** (pour écrire dedans, ou pour les afficher).
Pour faire ça, il suffit d'appliquer cette boucle for :

for(int i=0 ; i<nomTableau.length ; i++) ...

3 Les méthodes

3.1 Kesako ?

Présentation Les méthodes sont des **fonctions**. Elles permettent de décomposer le code du *main* en plusieurs sous-codes de manière logique.

Intérêt Le fait de créer des fonctions permet de les **réutiliser** facilement (calculer le double, ajouter une somme, ...).
Cela permet également de rendre le code **plus clair**, chaque méthode réalisant une tâche bien précise, il ne reste plus au final qu'à assembler les méthodes dans le bon ordre pour créer le programme.
Enfin, elles **simplifient** le code. Il est plus facile de travailler sur une méthode (un petit bout de code) que sur un gros code tout entier.

3.2 Déclarer une méthode

static **typeRetour** nomMethode(**parametres**) { ... }

typeRetour Il s'agit ici de mentionner de quel type sera la variable que la méthode va retourner (répondre) :

- **int** = entier
- **double** = réel
- **char** = caractère
- **void** = la méthode ne renvoie rien (elle réalise simplement une opération, mais ne parle pas)

parametres Il s'agit des variables dont la méthode à besoin lorsqu'elle est déclarée. Le nombre de paramètre n'est pas limité et peut être nul.
Les paramètres se précisent de la manière suivante (ici 2 variables) :

type nomVariable1, **type** nomVariable2

Au final, **déclarer un paramètre** = **initialiser une variable** pour la méthode.

Exemple Voici une méthode qui prend un **entier**, qui **calcule son double** et qui **renvoie** cette valeur :

```
static int calculerDouble(int nombre) {
  //On définit une variable de type int pour cette méthode
  int valeurFinale;

  //La variable enregistre la valeur souhaitée
  valeurFinale = nombre * 2;

  //La méthode renvoie la valeur (int) correspondante
  return valeurFinale;
}
```

3.3 Faire appel à une méthode

Une fois que la méthode est codée, il ne reste plus qu'à l'appeler par son petit nom au sein du *main* (ou dans une autre méthode si on en a besoin) :

```
//Exemple d'appel de la méthode calculerDouble dans le main

//On crée une variable entière
int varInt;

//On demande à l'utilisateur de donner un entier
System.out.println("Veuillez saisir une valeur entière");

//On affecte la valeur saisie par l'utilisateur à la variable
varInt = Lire.i();

//On affiche directement le résultat (la méthode renvoie la valeur)
System.out.println(varInt+" fois 2 : "+calculerDouble(varInt));

//On aurait également pu affecter le résultat de la méthode
//à une variable et ré-utiliser celle-ci.
```