Programme

- Introduction à la programmation informatique et aux différents langages de programmation.
- Présentation des concepts de base de la programmation de haut niveau. Aspect algorithmique. Introduction au langage Java.
- Syntaxe de base du langage Java, la sémantique et le concept de la compilation.
- Les variables, types et opérations mathématiques
- Références et affectation.
- Boucles et conditionnelles.
- Tableaux (accès aux composants, …).
- Procédures et Fonctions.
- Présentations de quelques classes clés.
- Des algorithmes fondamentaux.
- Des exemples applicatifs de l'ensemble des notions de Java présentées.

Les variables, types et opérations mathématiques

- Petit rappel: Catégories de poids
- Les différents types de variables
- Opérations arithmétiques
- Les conversions ou « cast »
- Exemples applicatives

Petit rappel: Catégories de poids (1/1)

- Il existe plusieurs catégories de poids de programme (Ko, Mo, Go, etc.).
- Tous ces poids correspondent au système métrique informatique.
- Le tableau suivant présente les poids les plus fréquemment rencontrés :

Raccourcis	Traduction	Correspondance
b	Bit	C'est la plus petite valeur informatique : soit 0 soit 1
o	Octet	regroupement de 8 bits, par exemple : 01011101
Ko	Kilo Octet	regroupement de 1024 octets
Mo	Mega Octet	regroupement de 1024 ko
Go	Giga Octet	regroupement de 1024 Mo
To	Tera Octet	regroupement de 1024 Go

Les différents types de variables (1/10)

Pour créer des variables dans la mémoire, il faut les déclarer. Une déclaration de variable se fait comme ceci :

<Type de la variable> <Nom de la variable> ;

- Cette opération se termine toujours par un point-virgule « ; ».
- Ensuite, on l'initialise en entrant une valeur.

Les différents types de variables (2/10)

En Java, nous avons deux types de variables :

1 - des variables de type simple ou « primitif » des nombres entiers, des nombres réels, des booléens ou encore des caractères.

2 - des variables de type complexe ou des « objets ».

Les différents types de variables (3/10)

Les variables de type numérique

- Le type byte (1 octet) peut contenir les entiers entre -128 et +127.

```
byte temperature;
temperature = 64;
```

- Le type short (2 octets) contient les entiers compris entre -32768 et +32767.

```
short vitesseMax;
vitesseMax = 32000;
```

Les différents types de variables (4/10)

Les variables de type numérique

- Le type int (4 octets) va de -2*109 à 2*109 (2 et 9 zéros derrière… ce qui fait déjà un joli nombre).

```
int temperatureSoleil;
temperatureSoleil = 15600000; //La température est exprimée en kelvins
```

- Le type long (8 octets) peut aller de -9×1018 à 9×1018 (encore plus gros…).

```
long anneeLumiere;
anneeLumiere = 9460700000000000L;
```

Afin d'informer la JVM que le type utilisé est **long**, vous **DEVEZ** ajouter un "**L**" à la fin de votre nombre, sinon le compilateur essaiera d'allouer ce dernier dans une taille d'espace mémoire de type entier et votre code ne compilera pas si votre nombre est trop grand…

Les différents types de variables (5/10)

Les variables de type numérique

Le type float (4 octets) est utilisé pour les nombres avec une virgule flottante.

```
float pi;
pi = 3.141592653f;
```

Ou encore:

```
float nombre;
nombre = 2.0f;
```

Vous remarquerez que nous ne mettons pas une virgule, mais un point ! Et vous remarquerez aussi que même si le nombre en question est rond, on écrit « .0 » derrière celui-ci, le tout suivi de « f ».

Les différents types de variables (6/10)

Les variables de type numérique

Le type double (8 octets) est identique à float, si ce n'est qu'il contient plus de chiffres derrière la virgule et qu'il n'a pas de suffixe.

```
double division;
division = 0.33333333333333333333333333333333333333333334d;
```

Ici encore, vous devez utiliser une lettre - le « d » - pour parfaire la déclaration de votre variable.

Les différents types de variables (7/10)

Des variables stockant un caractère

Le type `char` contient *un* caractère stocké entre apostrophes (« ' ' »), comme ceci :

```
char caractere;
caractere = 'A';
```

Des variables de type booléen

Le type `boolean`, lui, ne peut contenir que deux valeurs : `true` (vrai) ou `false` (faux), sans guillemets (ces valeurs sont natives dans le langage, il les comprend directement et sait les interpréter).

```
boolean question;
question = true;
```

Les différents types de variables (8/10)

Des variables de type String

- Le type String permet de gérer les chaînes de caractères, c'est-à-dire le stockage de texte.

- Il s'agit d'une variable d'un type plus complexe que l'on appelle objet.

```
//Première méthode de déclaration

String phrase;
phrase = "Titi et Grosminet";

//Deuxième méthode de déclaration
String str = new String();
str = "Une autre chaîne de caractères";

//Troisième méthode de déclaration
String string = "Une autre chaîne";

//Quatrième méthode de déclaration
String chaine = new String("Et une de plus !");
```

Attention :

- String commence par une majuscule !

- Lors de l'initialisation, on utilise des guillemets doubles (« " " »).

Les différents types de variables (9/10)

Convention de nommage : Il faut respecter cette convention:

- tous vos noms de classes doivent commencer par une majuscule ;
- tous vos noms de variables doivent commencer par une minuscule ;
- si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation ;
- tout ceci sans accentuation !

```
public class Toto{}

public class Nombre{}
public class TotoEtTiti{}
String chaine;
String chaineDeCaracteres;
int nombre;
int nombrePlusGrand;
```

Les différents types de variables (10/10)

Attention:

Veillez à bien respecter la casse (majuscules et minuscules), car une déclaration de `CHAR` à la place de `char` ou autre chose provoquera une erreur, tout comme une variable de type`string` à la place de `String` !

Astuce:

Lorsque nous avons plusieurs variables d'un même type, nous pouvons résumer tout ceci à une déclaration :

```
int nbre1 = 2, nbre2 = 3, nbre3 = 0;
```

Les opérateurs arithmétiques (1/8)

Les opérateurs arithmétiques sont:

- « + » : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères ; ne vous inquiétez pas, on aura l'occasion d'y revenir).
- « - » : permet de soustraire deux variables numériques.
- « * » : permet de multiplier deux variables numériques.
- « / » : permet de diviser deux variables numériques (mais je crois que vous aviez deviné).
- « % » : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le modulo.

Les opérateurs arithmétiques (2/8)

Quelques exemples de calcul - 1/2

```
int nbre1, nbre2, nbre3;

nbre1 = 1 + 3;
nbre2 = 2 * 6;
nbre3 = nbre2 / nbre1;
nbre1 = 5 % 2;
nbre2 = 99 % 8;
nbre3 = 6 % 3;
```

Les opérateurs arithmétiques (3/8)

Quelques exemples de calcul -2/2

```
int nbre1, nbre2, nbre3;  //Déclaration des variables

nbre1 = 1 + 3;           //nbre1 vaut 4
nbre2 = 2 * 6;           //nbre2 vaut 12
nbre3 = nbre2 / nbre1;   //nbre3 vaut 3
nbre1 = 5 % 2;           //nbre1 vaut 1, car 5 = 2 * 2 + 1
nbre2 = 99 % 8;          //nbre2 vaut 3, car 99 = 8 * 12 + 3
nbre3 = 6 % 3;           //là, nbre3 vaut 0, car il n'y a pas de reste
```

Les opérateurs arithmétiques (4/8)

D'autres exemples de calcul 1/2

```
int nbre1, nbre2, nbre3;

nbre1 = nbre2 = nbre3 = 0;

nbre1 = nbre1 + 1;
nbre1 = nbre1 + 1;
nbre2 = nbre1;
nbre2 = nbre2 * 2;
nbre3 = nbre2;
nbre3 = nbre3 / nbre3;
nbre1 = nbre3;
nbre1 = nbre1 - 1;
```

Les opérateurs arithmétiques (5/8)

D'autres exemples de calcul 2/2

```
int nbre1, nbre2, nbre3;    //Déclaration des variables

nbre1 = nbre2 = nbre3 = 0;  //Initialisation

nbre1 = nbre1 + 1;     //nbre1 = lui-même, donc 0 + 1 => nbre1 = 1
nbre1 = nbre1 + 1;     //nbre1 = 1 (cf. ci-dessus), maintenant, nbre1 = 1 + 1 = 2
nbre2 = nbre1;         //nbre2 = nbre1 = 2
nbre2 = nbre2 * 2;     //nbre2 = 2 => nbre2 = 2 * 2 = 4
nbre3 = nbre2;         //nbre3 = nbre2 = 4
nbre3 = nbre3 / nbre3; //nbre3 = 4 / 4 = 1
nbre1 = nbre3;         //nbre1 = nbre3 = 1
nbre1 = nbre1 - 1;     //nbre1 = 1 - 1 = 0
```

Les opérateurs arithmétiques (6/8)

Opération d'incrémentation

```
nbre1 = nbre1 + 1;

nbre1 += 1;
nbre1++;
++nbre1;
```

- Les trois premières syntaxes correspondent exactement à la même opération.
- La troisième est la plus utilisée, mais elle ne fonctionne que pour augmenter d'une unité la valeur de `nbre1` !
- Pour augmenter de 2 la valeur d'une variable, on utilise les deux premières syntaxes.
- La dernière fait la même chose que la troisième, mais il y a une subtilité dont nous reparlerons dans le chapitre sur les boucles.

Les opérateurs arithmétiques (7/8)

Opération de soustraction et décrémentation

Pour la soustraction, la syntaxe est identique :

```
nbre1 = nbre1 - 1;

nbre1 -= 1;
nbre1--;
--nbre1;
```

La troisième syntaxe s'appelle la **décrémentation**.

Les opérateurs arithmétiques (8/8)

Les raccourcis pour la multiplication

```
nbre1 = nbre1 * 2;

nbre1 *= 2;
nbre1 = nbre1 / 2;
nbre1 /= 2;
```

Très important : on ne peut faire du traitement arithmétique que sur des variables de même type sous peine de perdre de la précision lors du calcul. On ne s'amuse pas à diviser un `int` par un`float`, ou pire, par un `char` ! Ceci est valable pour tous les opérateurs arithmétiques et pour tous les types de variables numériques. Essayez de garder une certaine rigueur pour vos calculs arithmétiques.

Les conversions, ou « cast » (1/4)

Selon la nature de vos applications à développer, vous serez amenés à convertir des variables.

D'un type `int` en type `float` :

```
int i = 123;
float j = (float)i;
```

D'un type `int` en `double` :

```
int i = 123;
double j = (double)i;
```

Et inversement :

```
double i = 1.23;
double j = 2.9999999;
int k = (int)i;        //k vaut 1
k = (int)j;            //k vaut 2
```

Ce type de conversion s'appelle une « conversion d'ajustement », ou *cast* de variable.

Les conversions, ou « cast » (2/4)

Il est aussi possible de caster le résultat d'une opération mathématique en la mettant entre « ( ) » et en la précédant du type de cast souhaité. Donc :

```
double nbre1 = 10, nbre2 = 3;
int resultat = (int)(nbre1 / nbre2);
System.out.println("Le résultat est = " + resultat);
```

Pour bien faire, vous devriez mettre le résultat de l'opération en type `double`.

Et si on fait l'inverse : si nous déclarons deux entiers et que nous mettons le résultat dans un `double` ? Voici une possibilité :

```
int nbre1 = 3, nbre2 = 2;
double resultat = nbre1 / nbre2;
System.out.println("Le résultat est = " + resultat);
```

Vous aurez « 1 » comme résultat. Je ne caste pas ici, car un `double` peut contenir un `int`.

Les conversions, ou « cast » (3/4)

La priorité des opérations et le cast

Dans programme de code, l'affectation, le calcul, le cast, le test, l'incrémentation… toutes ces choses sont des opérations ! Et Java les fait dans un certain ordre, il y a des priorités.

Dans cet exemple:

```
int nbre1 = 3, nbre2 = 2;
double resultat = (double)(nbre1 / nbre2);
System.out.println("Le résultat est = " + resultat);
```

Dans le cas qui nous intéresse, il y a trois opérations :

- un calcul ; un cast sur le résultat de l'opération ;
- une affectation dans la variable `resultat`.

Java exécute notre ligne dans cet ordre ! Il fait le calcul (ici 3/2), il caste le résultat en `double`, puis il l'affecte dans notre variable `resultat`.

Les conversions, ou « cast » (4/4)

Dans l 'exemple précédent, pourquoi nous n'avons pas 1.5 ? …

```
int nbre1 = 3, nbre2 = 2;
double resultat = (double)(nbre1 / nbre2);
System.out.println("Le résultat est = " + resultat);
```

Pour avoir resultat=1,5: il faudrait caster chaque nombre avant de faire l'opération

```
int nbre1 = 3, nbre2 = 2;
double resultat = (double)(nbre1) / (double)(nbre2);
System.out.println("Le résultat est = " + resultat);
//affiche : Le résultat est = 1.5
```