Table des matières

1 Préambule

Java est un language orienté objet, il n'est pas possible d'y couper. Dans toute la suite, il sera supposé que vous connaissez déjà le principe de la Programmation Orienté Objet (POO).

En Java, l'exécution d'un programme va lancer la méthode **main()** de l'objet associé. Le code ci-dessous est le code minimum d'un programme Java qui défini la classe **sdz1**. Ce code ne fait rien.

```
public class sdz1 {

  /**
  * @param args
  */

  public static void main(String[] args) {
  // TODO Auto-generated method stub

  }
}
```

Le même code, affichant le sempiternel "Hello World" :

```
import java.util.Scanner;


public class sdz1 {

  /**
    * @param args
    */
  public static void main(String[] args) {
    // TODO Auto-generated method stub

    System.out.println("Hello World!");
  }

}
```

Quelques informations de base :

- Le programme commence par le lancement de la méthode **.main()** de la classe du programme principal.
- Les lignes doivent se terminer par "**;**"
- Il faut déclarer les variables avant de les utiliser (voir [§ 2.2, p3])
- Il faut compiler le programme avant de pouvoir l'utiliser (via une plateforme java, ce n'est pas un binaire mais un bytecode utilisable uniquement par un environnement Java)

2 Les bases

2.1 Commentaires

Il existe deux types de commentaires :

- les commentaires unilignes : introduits par les symboles //, ils mettent tous ce qui les suit en commentaires, du moment que le texte se trouve sur la même ligne que les //.

```
public static void main(String[] args){
  //Un commentaire
  //un autre
  //Encore un autre
  Ceci n'est pas un commentaire ! ! ! !
}
```

- les commentaires multilignes : ils sont introduits par les symboles /* et se terminent par les symboles */.

```
public static void main(String[] args){

  /*
  Un commentaire
  Un autre
  Encore un autre
  */
  Ceci n'est pas un commentaire ! !
}
```

2.2 Types de variables

En Java, nous avons deux type de variables :
- des variables de type simple ou "primitif",
- des variables de type complexe ou encore des objets.

Ce qu'on appelle des types simples, ou types primitifs, en Java ce sont tout bonnement des nombres entiers, des nombres réels, des booléens ou encore des caractères. Mais vous allez voir qu'il y a plusieurs façons de déclarer certains de ces types.

Commençons par 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;
```

- 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;
```

  C'est en kelvins...
- Le type *long*(8 octets) de -9*1018 à 9*1018 (encore plus gros...).

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

- Le type *float* (4 octets) correspond à des nombres avec virgule flottante.

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

  ou encore

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

  Vous remarquerez que nous ne mettons pas de virgule mais un point ! Et vous remarquerez aussi que même si le nombre en question est rond, on met tout de même .0 derrière celui-ci !
- Le type *double* (8 octets) est identique à float, si ce n'est qu'il contient un nombre plus grand derrière la virgule.

```
double division;
division = 0.33333333333333333333333333333333333333333333334;
```

Nous avons aussi des variables stockant du caractère

Le type *char* contient UN caractère stocké entre de simples quotes ' ' comme ceci...

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

Nous avons aussi le type booléen

Le type *boolean* qui lui contient **true** (vrai) ou **false** (faux).

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

Et aussi le type *String*. Celle-ci correspond à de la chaîne de caractères. Ici, il ne s'agit pas d'une variable mais d'un objet qui instancie une classe qui existe dans Java ; nous pouvons l'initialiser en utilisant l'opérateur unaire new() dont on se sert pour réserver un emplacement mémoire à un objet (mais nous reparlerons de tout ceci dans la partie deux, lorsque nous verrons les classes), ou alors lui affecter directement la chaîne de caractères.

Vous verrez que celle-ci s'utilise très facilement et se déclare comme ceci :

```
String phrase;
phrase = "Titi et gros minet";
//Deuxieme methode de declaration de type String
String str = new String();
str = "Une autre chaine de caracteres";
//La troisieme
String string = "Une autre chaine";
//Et une quatrieme pour la route
String chaine = new String("Et une de plus ! ");
```

On peut très bien compacter la phase de déclaration et d'initialisation en une seule phase ! Comme ceci :

```
int entier = 32;
float pi = 3.1416f;
char carac = 'z';
String mot = new String("Coucou");
```

Et lorsque nous avons plusieurs variables d'un même type, nous pouvons compacter tout ceci en une déclaration comme ceci :

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

2.3 Entrées/Sorties

2.3.1 Entrées

Afin de récupérer ce qu'on tape au clavier, il faut importer une nouvelle classe

```
import java.util.Scanner;
```

Voici l'instruction pour permettre à Java de récupérer ce que vous avez saisi et ensuite de l'afficher :

```
Scanner sc = new Scanner(System.in);
System.out.println("Veuillez saisir un mot :");
String str = sc.nextLine();
System.out.println("Vous avez saisi : " + str);
```

Dans le cas où on récupère autre chose qu'une chaîne de caractère, il faut vider la ligne via un `sc.nextLine();` avant de chercher à récupérer une chaîne de caractère.

```
Scanner sc = new Scanner(System.in);

System.out.println("Saisissez un entier : ");
int i = sc.nextInt();

System.out.println("Saisissez une chaine : ");
//On vide la ligne avant d'en lire une autre
sc.nextLine();
String str = sc.nextLine();
System.out.println("FIN ! ");
```

2.4 Boucles

2.4.1 Boucle if

```
if(//condition)
  {
  // execution des instructions si la condition est remplie


  }
else
  {
  // execution des instructions si la condition n'est pas remplie


  }
```

Exemple :

```
int i = 10;

if (i < 0)
  System.out.println("Le nombre est negatif");

else
  System.out.println("Le nombre est positif");
```

Remarque : On n'est pas obligés de mettre les accolades quand il n'y a qu'une seule ligne d'instruction dans la boucle.

On peut aussi mettre des tests multiples :

```
int i = 0;

if (i < 0)
  System.out.println("Ce nombre est negatif !");

else if(i > 0)
  System.out.println("Ce nombre est positif !!");

else
  System.out.println("Ce nombre est nul !!");
```

Si on souhaite faire beaucoup de tests, on peut souhaiter utiliser la structure *switch* à la place.

2.4.2 Boucle Switch

```
int nbre = 5;

switch (nbre)
{
  case 1:
    System.out.println("Ce nombre est tout petit");
    break;

  case 2:
    System.out.println("Ce nombre est tout petit");
    break;

  case 3:
```

```
        System.out.println("Ce nombre est un peu plus grand");
        break;

    case 4:
        System.out.println("Ce nombre est un peu plus grand");
        break;

    case 5:
        System.out.println("Ce nombre est la moyenne");
        break;

    case 6:
        System.out.println("Ce nombre est tout de meme grand");
        break;

    case 7:
        System.out.println("Ce nombre est grand");
        break;

    default:
        System.out.println("Ce nombre est compris entre 8 et 10");

}
```

2.4.3 Opérateur ternaire

La particularité des conditions ternaires réside dans le fait que trois opérandes (variable ou constante) sont mises en jeu mais aussi que ces conditions sont employées pour affecter des données dans une variable. Voici à quoi ressemble la structure de ce type de condition :

```
int x = 10, y = 20;
int max = (x < y) ? y : x ; //Maintenant max vaut 20
```

On peut faire par exemple :

```
int x = 10;
String type = (x % 2 == 0) ? "C' est pair" : "C' est impair" ;
//Ici type vaut "C' est pair"

x = 9;
type = (x % 2 == 0) ? "C' est pair" : "C' est impair" ;
//Ici type vaut "C' est impair"
```

2.4.4 Boucle while

```
int a = 1, b = 15;
while (a < b)
{
        System.out.println("coucou " +a+ " fois !!");
        a++;
}
```

On peut aussi faire :

```
//Une variable vide
String prenom;
// On initialise celle-ci a O pour oui !
char reponse = 'O';
//Notre objet Scanner, n'oubliez pas l'import de java.util.Scanner
Scanner sc = new Scanner(System.in);
//Tant que la reponse donnee est egale a oui
```

```
while (reponse == 'O')
{
  //On affiche une instruction
  System.out.println("Donnez un prenom : ");
  //On recupere le prenom saisi
  prenom = sc.nextLine();
  // On affiche notre phrase avec le prenom
  System.out.println("Bonjour " +prenom+ " comment vas-tu ?");
  //On demande si la personne veut faire un autre essai
  System.out.println("Voulez-vous reessayer ?(O/N)");
  //On recupere la reponse de l'utilisateur
  reponse = sc.nextLine().charAt(0);
}

System.out.println("Au revoir...");
//Fin de la boucle
```

2.4.5 Boucle do while

```
do{
  blablablablablablablabla
}while(a < b);
```

2.4.6 Boucle for

```
for(int i = 1; i <= 10; i++)
{
  System.out.println("Voici la ligne "+i);
}
```

On peut aussi boucler sur les éléments d'un tableau :

```
String tab[] = {"toto", "titi", "tutu", "tete", "tata"};

for(String str : tab)
   System.out.println(str);
```

Cette forme de boucle for est particulièrement adaptée au parcours de tableau. On peut naturellement se demander comment faire de même pour des tableaux multidimensionnels. La chose à retenir est que la variable en premier paramètre de la boucle for doit être du même type que la valeur de retour du tableau. Dans le cas d'un tableau multi-dimensionnel, cette dernière sera un tableau de dimension inférieure. En conséquence, on peut boucler sur des sous tableaux, puis sur les éléments de ces derniers via des boucles imbriquées :

```
String tab[][] = {{"toto", "titi", "tutu", "tete", "tata"},
                  {"1", "2", "3", "4"}};
int i = 0, j = 0;

for(String sousTab[] : tab)
{
  i = 0;
  for(String str : sousTab)
  {
    System.out.println("La valeur de la nouvelle boucle est : " + str);
    System.out.println("La valeur du tableau a l'indice ["
      + j + "][" +i +"] est : " +  tab[j][i] + "\n");
    i++;
  }
  j++;
}
```

2.5 Tableaux

On définit des tableaux de la même manière que les éléments qui le constituent. Un tableau a donc un type associé et ne peut stocker que des éléments de ce type là.

Pour définir un tableau sans l'initialiser on fait :

```
int tableauEntier[] = new int[6];
//ou encore
int[] tableauEntier2 = new int[6];
```

mais la définition d'un tableau initialisé se fait elle de la façon suivante :

```
String tableauChaine[] = {"chaine1", "chaine2", "chaine3" , "chaine4"};
```

On peut définir des tableaux multi-dimensionnels :

```
int premiersNombres[][] = { {0,2,4,6,8},{1,3,5,7,9} };
```

Nous voyons bien ici les deux lignes de notre tableau symbolisées par les doubles crochets `[][]`. Ce genre de tableau n'est rien d'autre que plusieurs tableaux en un. Ainsi, pour passer d'une ligne à l'autre, nous jouerons avec la valeur du premier crochet.

Exemple : `premiersNombres[0][0]` correspondra au premier élément de la colonne paire.
Et `premiersNombres[1][0]` correspondra au premier élément de la colonne impaire.

2.5.1 Propriétés

La longueur d'un tableau **tab** est donnée par :

```
tab.length
```

Index