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.

```
1  public class sdz1 {
2
3    /**
4    * @param args
5    */
6
7    public static void main(String[] args) {
8    // TODO Auto-generated method stub
9
10   }
11 }
```

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

```
1  import java.util.Scanner;
2
3
4  public class sdz1 {
5
6    /**
7      * @param args
8      */
9    public static void main(String[] args) {
10     // TODO Auto-generated method stub
11
12     System.out.println("Hello World!");
13   }
14
15 }
```

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 //.

```
1 public static void main(String[] args){
2   //Un commentaire
3   //un autre
4   //Encore un autre
5   Ceci n'est pas un commentaire ! ! ! !
6 }
```

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

```
1 public static void main(String[] args){
2
3   /*
4   Un commentaire
5   Un autre
6   Encore un autre
7   */
```

```
8     Ceci n'est pas un commentaire ! !
9  }
```

Il existe en fait un troisième type de commentaires : les *javadocs*. Ce sont des commentaires avec une mise en forme particulière (ils commencent par `/**` au lieu de `/*`) qui vont permettre de générer de la documentation de manière automatisée. En voici un exemple :

```
1  /**
2   * Retourne une chaine de caracteres selon le resultat de la comparaison
3   * @param v1
4   *             objet Ville
5   * @return comparaison de deux ville
6   */
```

Dans *Eclipse*, il vous suffit d'aller dans **Project** et de cliquer sur **Generate Javadoc**.

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.

```
1 byte temperature;
2 temperature = 64;
```

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

```
1 short vitesseMax;
2 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).

```
1 int temperatureSoleil;
2 temperatureSoleil = 15600000;
```

  C'est en kelvins...

- Le type *long*(8 octets) de -9*1018 à 9*1018 (encore plus gros...).

```
1 long anneeLumiere;
2 anneeLumiere = 9460700000000000;
```

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

```
1 float pi;
2 pi = 3.141592653f;
```

  ou encore

```
1 float nombre;
2 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.

```
1 double division;
2 division = 0.33333333333333333333333333333333333333333333334;
```

Nous avons aussi des variables stockant du caractère

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

```
1 char caractere;
2 caractere = 'A';
```

Nous avons aussi le type booléen

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

```
1 boolean question;
2 question = true;
```

2.2.1 Chaînes de caractères

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 ! ");
```

String étant un objet, il possède des méthodes afin de les manipuler. En voici quelques exemples :

- *toLowerCase()*
  Cette méthode permet de transformer toute saisie clavier de type caractère en minuscules. Elle n'a aucun effet sur les nombres, puisqu'ils ne sont pas assujettis à cette contrainte. Vous pouvez donc utiliser cette fonction sur une chaîne de caractères comportant des nombres. Elle s'utilise comme ceci :

  ```
  String chaine = new String("COUCOU TOUT LE MONDE !");
  String chaine2 = new String();
  chaine2 = chaine.toLowerCase();//donne "coucou tout le monde !"
  ```

- *toUpperCase()*
  Celle-là est facile, puisqu'il s'agit de l'opposée de la précédente. Elle transforme donc une chaîne de caractères en majuscules. Et s'utilise comme suit :

  ```
  String chaine = new String("coucou coucou"), chaine2 = new String();
  chaine2 = chaine.toUpperCase();//donne "COUCOU COUCOU"
  ```

- *concat()*
  Très explicite, celle-là permet de concaténer deux chaînes de caractères.

  ```
  String str1 = new String("Coucou "), str2 = new String("toi !");
  String str3 = new String();
  str3 = str1.concat(str2);//donne "Coucou toi !"
  ```

- *length()*
  Celle-là permet de donner la longueur d'une chaîne de caractères (en comptant les espaces blancs).

  ```
  String chaine = new String("coucou ! ");
  int longueur = 0;
  longueur = chaine.length();//donne 9
  ```

- *equals()*
  Permet de voir si deux chaînes de caractères sont identiques. Donc, de faire des tests. C'est avec cette fonction que vous ferez vos tests de conditions, lorsqu'il y aura des *String*. Exemple concret :

  ```
  String str1 = new String("coucou"), str2 = new String("toutou");

  if (str1.equals(str2))//Si les deux chaines sont identiques
          System.out.println("Les deux chaines sont identiques !");

  else
          System.out.println("Les deux chaines sont differentes !");
  ```

  Vous pouvez aussi demander la non vérification de l'égalité grâce à l'opérateur de négation « ! », ce qui nous donne :

  ```
  String str1 = new String("coucou"), str2 = new String("toutou");

  if (!str1.equals(str2))//Si les deux chaines sont differentes
          System.out.println("Les deux chaines sont differentes !");

  else
          System.out.println("Les deux chaines sont identiques !");
  ```

Le principe de ce genre de condition fonctionne de la même façon pour les boucles. Et dans l'absolu, cette fonction retourne un booléen. C'est pourquoi nous pouvons utiliser cette fonction dans les tests de condition.

```
String str1 = new String("coucou"), str2 = new String("toutou");
boolean Bok = str1.equals(str2);//ici Bok prendra la valeur false
```

– *charAt()*
Le résultat de cette méthode sera un caractère, car il s'agit d'une méthode d'extraction de caractères, je dirais même d'UN caractère. Elle ne peut s'opérer que sur des *String* ! Elle possède la même particularité que les tableaux, c'est-à-dire que, pour cette méthode, le premier caractère sera le numéro 0. Cette méthode prend un entier comme argument.

```
String nbre = new String("1234567");
char carac = ' ';
carac = nbre.charAt(4);//renverra ici le caractere 5
```

– *substring()*
Comme son nom l'indique, elle permet d'extraire une sous-chaîne de caractères d'une chaîne de caractères. Cette méthode prend 2 entiers comme arguments. Le premier définit le début de la sous-chaîne à extraire inclus, le deuxième correspond au dernier caractère à extraire exclus. Et le premier caractère est aussi le numéro 0.

```
String chaine = new String("la paix niche"), chaine2 = new String();
chaine2 = chaine.substring(3,13);//permet d'extraire "paix niche"
```

– *indexOf()*/*lastIndexOf()*
indexOf() permet d'explorer une chaîne de caractères depuis son début. *lastIndexOf()* depuis sa fin, mais renvoie l'index depuis le début de la chaine. Elle prend un caractère, ou une chaîne de caractères comme argument, et renvoie un int. Tout comme *charAt()* et *substring()*, le premier caractère est à la place 0. Je crois qu'ici un exemple s'impose, plus encore que pour les autres fonctions :

```
String mot = new String("anticonstitutionnellement");
int n = 0;

n = mot.indexOf('t');       // n vaut 2
n = mot.lastIndexOf('t');         // n vaut 24
n = mot.indexOf("ti");     // n vaut 2
n = mot.lastIndexOf("ti");       // n vaut 12
n = mot.indexOf('x');       // n vaut -1
```

2.2.2 Astuces

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;
```

On peut aussi convertir une variable ou une formule dans un autre type de donnée via ce qu'on appelle un **cast** :

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


Il peut y avoir des pertes de précision au sein même des opérations mathématiques que la conversion du résultat via un cast n'empêchera pas. Ex :

```
int i = 123, j = 5;
double k = (double) (i / j);
```

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 !!");
```

```
8
9  else
10   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

```
1  int nbre = 5;
2
3  switch (nbre)
4  {
5    case 1:
6      System.out.println("Ce nombre est tout petit");
7      break;
8
9    case 2:
10     System.out.println("Ce nombre est tout petit");
11     break;
12
13   case 3:
14     System.out.println("Ce nombre est un peu plus grand");
15     break;
16
17   case 4:
18     System.out.println("Ce nombre est un peu plus grand");
19     break;
20
21   case 5:
22     System.out.println("Ce nombre est la moyenne");
23     break;
24
25   case 6:
26     System.out.println("Ce nombre est tout de meme grand");
27     break;
28
29   case 7:
30     System.out.println("Ce nombre est grand");
31     break;
32
33   default:
34     System.out.println("Ce nombre est compris entre 8 et 10");
35
36 }
```

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 :

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

On peut faire par exemple :

```
1  int x = 10;
2  String type = (x % 2 == 0) ? "C' est pair" : "C' est impair" ;
3  //Ici type vaut "C' est pair"
4
5  x = 9;
6  type = (x % 2 == 0) ? "C' est pair" : "C' est impair" ;
7  //Ici type vaut "C' est impair"
```

2.4.4 Boucle while

```
1  int a = 1, b = 15;
2  while (a < b)
3  {
4           System.out.println("coucou " +a+ " fois !!");
5           a++;
6  }
```

On peut aussi faire :

```
//Une variable vide
String prenom;
// On initialise celle-ci a 0 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
```

3 Les classes

Toute classe possède un constructeur (une méthode particulière, lancée lors de l'initialisation d'une instance de classe), ayant le même nom que la classe elle-même.

Prenons un exemple avec une classe **Ville** :

```
public class Ville{
  /**
  * Stocke le nom de notre ville
  */
  String nomVille;
  /**
  * Stocke le nom du pays de notre ville
  */
  String nomPays;
  /**
  * Stocke le nombre d'habitants de notre ville
  */
  int nbreHabitant;

  /**
   * Constructeur par defaut
   */
  public Ville(){
    System.out.println("Creation d'une ville !");
    nomVille = "Inconnu";
    nomPays = "Inconnu";
    nbreHabitant = 0;
  }

}
```

Remarque : Il est possible de surcharger le constructeur et ainsi avoir plusieurs constructeurs en fonction des paramètres passés lors de l'initialisation de l'instance de classe.

Dans l'exemple précédent, les variables de classes sont publiques et modifiables directement. La philosophie de la POO est d'utiliser des *accesseurs* getVar (qui renvoient la valeur d'une variable var) et des *mutateurs* setVar (qui modifient la valeur de la variable var). En conséquence, on mettra plutôt les variables avec une portée privée, et on créera des méthodes permettant d'afficher ou modifier ces dernières. La classe devient alors :

```java
public class Ville {
  /**
  * Stocke le nom de notre ville
  */
  private String nomVille;
  /**
  * Stocke le nom du pays de notre ville
  */
  private String nomPays;
  /**
  * Stocke le nombre d'habitants de notre ville
  */
  private int nbreHabitant;

  /**
  * Constructeur par defaut
  */
  public Ville(){
    System.out.println("Creation d'une ville !");
    nomVille = "Inconnu";
    nomPays = "Inconnu";
    nbreHabitant = 0;
  }

  /**
  * Constructeur d'initialisation
  * @param pNom
  *                       Nom de la Ville
  *  @param pNbre
  *                       Nombre d'habitants
  *  @param pPays
  *                       Nom du pays
  */
  public Ville(String pNom, int pNbre, String pPays)
  {
    System.out.println("Creation d'une ville avec des parametres !");
    nomVille = pNom;
    nomPays = pPays;
    nbreHabitant = pNbre;
  }

 //*****************************************************************************
  //                                        ACCESSEURS
  //*****************************************************************************

  /**
   * Retourne le nom de la ville
   * @return le nom de la ville
   */
  public String getNom()
  {
    return nomVille;
  }

  /**
   * Retourne le nom du pays
   * @return le nom du pays
   */
  public String getNomPays()
  {
    return nomPays;
  }

  /**
   * Retourne le nombre d'habitants
   * @return nombre d'habitants
   */
 public int getNombreHabitant()
 {
    return nbreHabitant;
 }

 //*****************************************************************************
```

```
74  //                                          MUTATEURS
75  //*************************************************************************
76
77  /**
78   * Definit le nom de la ville
79   * @param pNom
80   *              nom de la ville
81   */
82  public void setNom(String pNom)
83  {
84     nomVille = pNom;
85  }
86
87  /**
88   * Definit le nom du pays
89   * @param pPays
90   *              nom du pays
91   */
92  public void setNomPays(String pPays)
93  {
94     nomPays = pPays;
95  }
96
97  /**
98   * Definit le nombre d'habitants
99   * @param nbre
100  *              nombre d'habitants
101  */
102  public void setNombreHabitant(int nbre)
103  {
104    nbreHabitant = nbre;
105  }
106
107 }
```

La création d'une instance de classe se fait via le mot clé **new** :

```
1 Ville bordeaux = new Ville();
```

Il y a deux principaux types de variables. Tout d'abord les variables de classes, qui sont définies dans la classe, avec l'attribut *static*. Cette variable sera commune à toutes les instances de la classe.

Remarque : Pour accéder à une *variable de classe* **var** (de la classe **Ville** par exemple) à l'intérieur même de celle-ci (ou une de ses instances), on y accède via :

```
1 Ville.var
```

```
1 public class Ville {
2
3   /**
4    * Variable de classe
5    */
6   static String nomVille;
```

Il y a ensuite les variables d'instance, qui sont définies dans la classe. Cette variable **var** est accessible, à l'intérieur de la classe via **this.var** où *this* désigne l'instance de classe.

Remarque : En effet, en règle générale, on appelle des méthodes ou des paramètres grâce à notre variable qui nous sert de référence, celle-ci suivie de l'opérateur ".", puis du nom de la dite méthode/-variable.

```
1 String str = new String("opizrgpinbzegip");
2 str = str.subString(0,4);
```

Mais pour accéder aux données ou méthode de notre objet à l'intérieur même de celui-ci, cette méthode n'est pas possible. Pour y remédier, il faut utiliser le mot-clé *this* :

```java
public class Ville {

   /**
   * Variable d'instance
   */
   String nomVille;

   public Ville(String nom) {
      this.nomVille = "nom";
   }
}
```

On peut créer des variables privées (notamment pour faire des variables locales) en rajoutant un attribu privé à une variable. Il est recommandé de définir ces variables avec l'attribut *private* et de créer des *accesseurs* et des *mutateurs* afin de voir le contenu de la variable et de le modifier.

3.1 Générer automatiquement les accesseurs et les mutateurs avec Eclipse

Une fois qu'on a créé une classe avec beaucoup de variables d'instances ou de classe, *eclipse* permet de générer automatiquement les *accesseurs* et les *mutateurs*.

Il faut pour cela aller dans le menu **Source**, puis l'option générale **Generate Getters and Setters**. *Eclipse* propose alors la liste des variables présentes dans la classe courante.

3.2 Méthodes d'instance

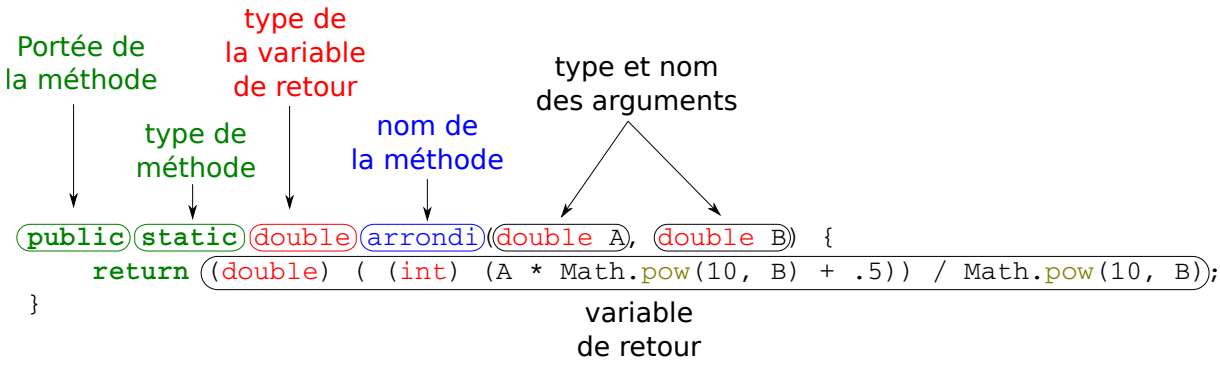

FIGURE 1 – Explication des différents attributs d'une méthode. Le but ici est de présenter à quoi correspond la syntaxe, sans présenter les différents choix possibles.

Les méthodes ne sont pas limitées en nombre de paramètres (s'il n'y a pas d'argument, il faut au minimum « `String[] args` ».

Il existe trois grands types de méthodes :

- celles qui ne renvoient rien. Elles sont de type *void*. Ces types de méthodes n'ont pas d'instruction **return** !
- celles qui retournent des types primitifs (*double*, *int*...). Elles sont de type *double*, *int*, *char*... Celles-ci ont une instruction **return**.
- celles qui retournent des objets. Par exemple, une méthode qui retourne un objet de type *String*. Celles-ci aussi ont une instruction **return**.

Remarque : On n'imbrique pas les méthodes et elles doivent toutes faire partie d'une classe.

⚠ Les méthodes de la classe **main**, c'est à dire la classe lancée par défaut au début du programme et qui contient le *main()*, doivent être *static*

3.3 Méthode de classe

Toutes les **méthodes** d'une **classe** n'utilisant que des *variables de classes* doivent être déclarées *static* ! On les appelle des **méthodes de classes** car elles sont globales à toutes vos instances !

Remarque : Par contre ceci n'est plus vrai si une méthode utilise des variables d'instances et des variables de classes...

L'*accesseur* d'une *variable de classe* doit aussi être déclarée *static*, c'est une règle !

3.4 La surcharge de méthode

La surcharge de méthode consiste à garder un nom de méthode (donc un type de traitement à faire, pour nous, lister un tableau) et de changer la liste ou le type de ses paramètres.

Nous allons surcharger notre méthode afin qu'elle puisse travailler avec des **int** par exemple :

```
1  static void parcourirTableau(String[] tab)
2    {
3       for(String str : tab)
4          System.out.println(str);
5    }
6
7  static void parcourirTableau(int[] tab)
8    {
9       for(int str : tab)
10         System.out.println(str);
11   }
```

On peut aussi faire de même avec les tableaux à 2 dimensions ou ajouter des paramètres à la méthode.

Index