

Table des matières

1	Préambule	2
2	Les bases	2
2.1	Commentaires	2
2.2	Types de variables	3
2.2.1	Chaînes de caractères	4
2.2.2	Astuces	5
2.3	Entrées/Sorties	6
2.3.1	Entrées	6
2.4	Boucles	6
2.4.1	Boucle if	6
2.4.2	Boucle Switch	7
2.4.3	Opérateur ternaire	8
2.4.4	Boucle while	8
2.4.5	Boucle do while	9
2.4.6	Boucle for	9
2.5	Tableaux	9
2.5.1	Propriétés	10
3	Les classes	10
3.1	Méthodes de classe	13
3.2	La surcharge de méthode	13
	Index	15

1 Préambule

Java est un langage 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 définit 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 `*/`.


```
boolean question;
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 = "Titiet_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 chaîne. 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 chaîne:");
//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{
    blablablablablablabla
}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 défaut
     */
    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 :

```

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 default
     */
    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;
    }

    /**
     * *****
     * ***** MUTATEURS *****
     * *****
     */

    /**
     * Definit le nom de la ville
     * @param pNom
     *          nom de la ville
     */
    public void setNom(String pNom)
    {
        nomVille = pNom;
    }

    /**
     * Definit le nom du pays
     * @param pPays
     *          nom du pays
     */
    public void setNomPays(String pPays)
    {
        nomPays = pPays;
    }

    /**
     * Definit le nombre d'habitants
     * @param nbre
     *          nombre d'habitants
     */
    public void setNombreHabitant(int nbre)
    {
        nbreHabitant = nbre;
    }
}

```

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

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

Il y a deux principaux types de variables :

- 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.

```
public class Ville {
```

```
/**
 * Variable de classe
 */
static String nomVille;
```

- 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 (c'est un concept de POO difficile à saisir, car c'est abstrait, ça désigne la variable instanciée, que l'on ne connaît pas *a priori* lorsque l'on programme. En gros, si je défini une variable toto, de type « la classe considérée », **this** désigne, DANS la variable toto, l'instance toto de la classe.

```
public class Ville {

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

On peut créer des variables privées (notamment pour faire des variables locales) en rajoutant un attribut privé à une variable.

3.1 Méthodes de classe

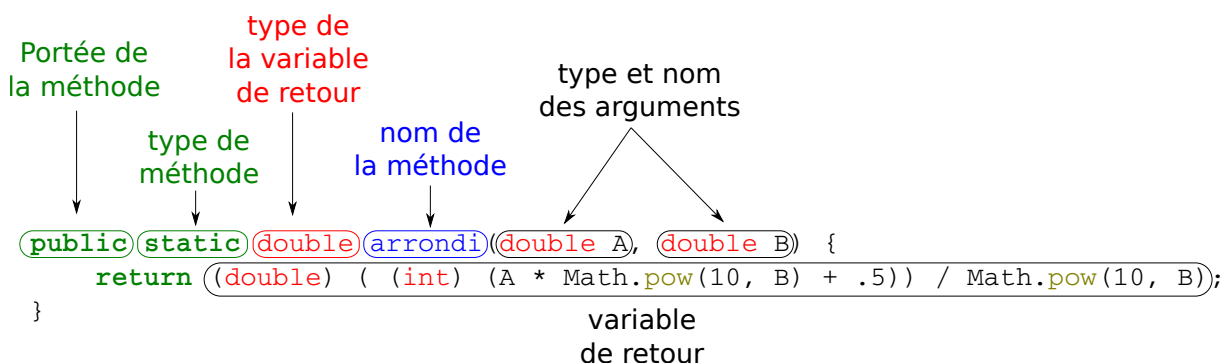


FIGURE 1 – Explication des différents attributs d'une méthode. Le but ici est de présenter à quoi correspond la syntaxe, sans présenter les différents choix possibles.

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`.

3.2 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 :

```
static void parcourirTableau(String [] tab)
{
    for(String str : tab)
        System.out.println(str);
}

static void parcourirTableau(int [] tab)
{
    for(int str : tab)
        System.out.println(str);
}
```

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

Index

boucle
 do while, 9
 for, 9
 if, 6
 switch, 7
 while, 8

charAt(), 5
concat(), 4

equals(), 4

indexOf(), 5

lastIndexOf(), 5
length(), 4

main(), 10

static, 10
substring(), 5

tableaux, 9
toLowerCase(), 4
toUpperCase(), 4

type
 boolean, 3
 byte, 3
 char, 3, 10
 double, 3, 10
 float, 3
 int, 3, 10
 long, 3
 short, 3
 String, 4, 5, 10

void, 10