M2 LITL - U.E. Programmation pour le TAL
Aide-mémoire Java

Franck Sajous - CLLE-ERSS

Ce document est disponible à l'adresse : `http://fsajous.free.fr/`

1 Classes et objets : membres et méthodes

1.1 Définition de classe

Listing 1 – Définition de classe

```
1 package nom_paquet;
2
3 public class MaClasse
4 {
5   // définition de la classe
6 }
```

1.2 Définition des membres

Syntaxe : `accès type nom [= valeur];`

- accès = `public`, `private` ou `protected`
  - `public` : aucune restriction d'accès en consultation ni en modification
  - `private` : accès en consultation et modification uniquement au sein de la classe
  - `protected` : accès en consultation et modification uniquement au sein de la classe et des classes dérivées

  Note : le comportement de ces modifieurs d'accès peut être légèrement différent (e.g. accessibilité au sein d'un paquetage). Nous n'entrerons pas dans ces détails.
- type : type primitif (`int`, `double`, `boolean`, etc.) ou nom de classe
- nom : nom du membre, commençant nécessairement par une lettre (minuscule, par convention), puis formé de lettres, chiffres et du caractère underscore _
- initialisation facultative (permet de donner une valeur initiale à un membre)

Listing 2 – Définition des membres

```
1 private   int      nbEntier;
2 private   int      autreNombreEntier = 23;
3 public    double   nombreReel;
4 public    String   maChaine;
5 protected String   monAutreChaine = "ma jolie chaine";
```

```
6  private    Color     maCouleur;
7  private    Color     monAutreCouleur = new Color ();
```

1.3 Définition de méthodes

Syntaxe : `accès [static] type_retour nom_méthode ([liste paramètres])`

- accès : `public`, `private` ou `protected`, spécifie si la méthode peut être invoquée uniquement depuis l'intérieur de la classe, depuis une autre classe ou depuis les classes dérivées
- static : précise s'il s'agit d'une méthode statique ou non
- type_retour : type (primitif ou classe) de la valeur retournée par la méthode. Si la méthode ne retourne aucune valeur, on écrit `void`.
- liste paramètres (facultative) : liste des paramètres que l'on passe à la méthode en l'invoquant. Pour chaque paramètre, on précise un type et un nom. Les paramètres sont séparés par des virgules.

Exemple : voir le listing 4

1.4 Point d'entrée : méthode principale (d'une classe donnée)

Listing 3 – Méthode `main()`

```
1  public static void main (String args[])
2  {
3    // corps de la méthode
4  }
```

1.5 Exemple

Listing 4 – Exemple de définition de classe

```
1  public class Token
2  {
3    private final static int STEM_LENGTH = 7;
4    private String wordForm;
5    private String lemma;
6    private String POS;
7
8    public String getWordForm()
9    {
10     return wordForm;
11   }
12
13   public void setWordForm(String wf)
14   {
15     wordForm = wf;
16   }
17
18   public void setLemma (String l)
19   {
```

```
20      lemma = l;
21    }
22
23    public String getLemma ()
24    {
25      return lemma;
26    }
27
28    public String getPOS()
29    {
30      return POS;
31    }
32
33    public void setPOS(String pOS)
34    {
35      POS = pOS;
36    }
37
38    public String getStem ()
39    {
40      if (wordForm.length() > STEM_LENGTH)
41        return wordForm.substring(0, STEM_LENGTH);
42      else
43        return wordForm;
44    }
45
46    public void displayWithLemma ()
47    {
48      System.out.println(wordForm + "\t" + POS + "\t" + lemma);
49    }
50
51    public void displayWithStem ()
52    {
53      System.out.println(wordForm + "\t" + POS
54                                  + "\t" + getStem());
55    }
56
57    public void displayBoth ()
58    {
59      displayWithLemma();
60      displayWithStem();
61    }
62  }
```

Listing 5 – Exemple d'utilisation de la classe

```
1   public class TokenTest
2   {
3     public static void main(String[] args)
4     {
5       Token myTok = new Token();
6
7       myTok.setWordForm ("encapsuleriez");
8       myTok.setLemma("encapsuler");
```

```
9       myTok.setPOS ("V");
10
11      System.out.println("form: " + myTok.getWordForm());
12
13      myTok.displayBoth();
14    }
15
16  }
```

2 Types primitifs, chaînes de caractères

2.1 Entiers : int

Type qui représente les nombres entiers : $-n, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, n$

`+, -`
`int nb = 3 - 2;`
`nb = nb - 1;`
	addition, soustraction

`*` `int nbMult = nb * 3;`	multiplication
`/` `nb = 3 / 2;`	division entière nb vaut 1
`%` `nb = 11 % 3;`	modulo (reste de division entière) nb vaut 2

`++, --`
`nb++;`
`nb--;`
	auto-incrément, auto-décrément
	incrémente nb (équivalent à `nb = nb + 1;`)
	décrémente nb (équivalent à `nb = nb - 1;`)

`*=`
`nb *= 5;`
`/=`
`nb /= 5;`
	multiplication
	équivalent à `nb = nb * 5;`
	division entière
	équivalent à `nb = nb / 5;`

`==` nb == 2	égalité `true` si nb vaut 2, `false` sinon
`!=` nb != 2	différence `true` si nb est différent de 2, `false` sinon
`<, >` nb < 2 nb > 2	inférieur à, supérieur à `true` si nb est inférieur à 2, `false` sinon `true` si nb est supérieur à 2, `false` sinon
`<=, >=` nb <= 2 nb >= 2	inférieur ou égal à, supérieur ou égal à `true` si nb est inférieur ou égal à 2, `false` sinon `true` si nb est supérieur ou égal à 2, `false` sinon

2.2 Réels : double

Type qui représente les nombres réels : $-3.2, 5.0, 22/7, -3.333333\ldots$, *etc.*

`+, -, *`	addition, soustraction et multiplication même syntaxe et même sémantique que pour les entiers
`/` `double nb = 3 / 2;`	division réelle nb vaut 1.5

`*=`
`nb *= 5;`
`/=`
`nb /= 5;`
	multiplication
	équivalent à `nb = nb * 5;`
	division réelle
	équivalent à `nb = nb / 5;`

Opérateurs de comparaison `==, !=, <, >, <=, >=` : même syntaxe et même sémantique que pour les entiers.

2.3 Booléens : boolean

Type à deux valeurs : `true` et `false`

`&&, ||`
`a && b`
`a || b`
	ET logique, OU logique
	vrai si a est vrai ET b est vrai
	vrai si a est vrai OU b est vrai

`!` `!expression`	négation vrai si expression vaut false, faux si expression vaut true

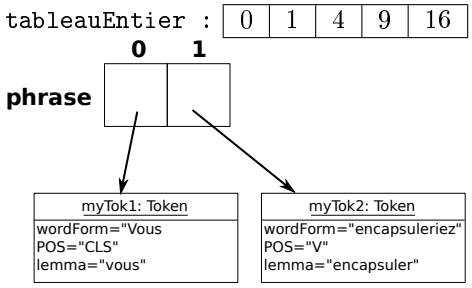
3 Tableaux

Syntaxe :
`type nomtableau [];`
`type nomtableau [] = new type[taille];`
- type : type des éléments que contiendra le tableau.
- taille : nombre d'éléments maximum que peut contenir le tableau.
- indice du premier élément du tableau : 0
- taille du tableau : `nom_tableau.length` (de type `int`)
- indice du dernier élément du tableau : `nom_tableau.length - 1`

Listing 6 – Tableaux

```
1  int tableauEntiers[];
2
3  tableauEntiers = new int[5];
4
5  for (int i = 0; i < tableauEntiers.length; i++)
6      tableauEntiers[i] = (i * i);
7
8  for (i = 0; i < tableauEntiers.length; i++)
9      System.out.println ("Le carré de " + String.valueOf(i)
10         + " est " + String.valueOf(tableauEntiers[i]));
11
12 // initialisation lors de la déclaration
13 // taille non précisée car = au nombre d'éléments
14 double tableauFlottants[] = {3.2, 5.4, -6.458};
15 System.out.println (tableauFlottants.length); // affiche : 3
16
```

```
17 // tableau d'objets
18
19 Token myTok1 = new Token();
20
21 myTok1.setWordForm ("Vous");
22 myTok1.setLemma("vous");
23 myTok1.setPOS ("CLS");
24
25 Token myTok2 = new Token();
26
27 myTok2.setWordForm ("encapsuleriez");
28 myTok2.setLemma("encapsuler");
29 myTok2.setPOS ("V");
30
31 Token[] phrase = new Token[2];
32 phrase[0] = myTok1;
33 phrase[1] = myTok2;
34
35 for (int i = 0; i < phrase.length; i++)
36 {
37   phrase[i].displayWithStem ();
38 }
```



4 Chaînes de caractères : la classe `String`

On peut considérer une chaîne de caractères comme étant un tableau de n caractères. La classe `String` peut être vue comme une surcouche de cette représentation, qui offre des fonctionnalités permettant de manipuler facilement des chaînes.
Quelques méthodes de la classe `String` :

`char`	`charAt(int index)` retourne le caractère situé à l'indice spécifié
`boolean`	`contains(CharSequence s)` retourne true si la chaîne contient la sous-chaîne `s` (`CharSequence` est une interface implémentée par `String`), false sinon
`boolean`	`endsWith(String suffix)` retourne true si la chaîne se termine par `suffix`, false sinon.
`boolean`	`startsWith(String prefix)` retourne true si la chaîne commence par `prefix`, false sinon.
`boolean`	`isEmpty()` retourne true si la chaîne est vide, false sinon.
`int`	`length()` retourne la longueur (nombre de caractères) de la chaîne. Attention, contrairement aux tableaux dont `length` est un membre, dans la classe `String`, `length()` est une méthode.
`boolean`	`equals(Object obj)` Si obj est de type `String`, retourne vrai si les deux chaînes ont la même valeur et false sinon. Sinon, renvoie true/false en comparant la valeur de la chaîne et la valeur retournée par la méthode `toString()` de `obj`.
`boolean`	`equalsIgnoreCase(String s)` Renvoie true si la chaîne est identique à `s` sans tenir compte de la casse (différence majuscules/minuscules) et false sinon.

Cette liste n'est pas exhaustive : consultez la documentation de l'API, notamment pour les méthodes `replace()`, `substring()`, `toUppperCase()`, `toLowerCase()`, `indexOf()`...