LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

II - LES OPERATEURS

1. Opérateurs arithmétiques

0PERATEURS	SIGNIFIATIONS
+	Addition
-	Soustratction
*	Multiplication
%	modulo
++	Incrémentation
--	Décrémentation
/	Division

2. Opérations bit à bit sur les entiers

0PERATEURS	SIGNIFIATIONS
&	ET
|	OU
^	XOR
~	COMPLEMENT
<<	DECALAGE A GAUCHE
>>	DECALAGE A DROITE

LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

II - LES OPERATEURS

3. Opérations d'affectation

Le signe = est l'opérateur d'affectation et s'utilise avec une expression de la forme :

variable = expression

EX: produit = $variable1;

Opérateur	Exemple	Signification
=	a=10	équivalent à : a = 10
+=	a+=10	équivalent à : a = a + 10
-=	a-=10	équivalent à : a = a – 10
*=	a*=10	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	a%=10	reste de la division
^=	a^=10	équivalent à : a = a ^ 10
<< =	a<<=10	équivalent à : a = a << 10 a est complété par des zéros à droite
>>=	a>>=10	équivalent à : a = a >> 10 a est complété par des zéros à gauche
>>>=	a>>>=10	équivalent à : a = a >>> 10 décalage à gauche non signé

LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

II - LES OPERATEURS

4. Opérations de comparaisons

Opérateur	Exemple	Signification
>	a > 10	strictement supérieur
<	a < 10	strictement inférieur
>=	a >= 10	supérieur ou égal
<=	a <= 10	inférieur ou égal
==	a == 10	Egalité
!=	a != 10	diffèrent de
&	a & b	ET binaire
^	a ^ b	OU exclusif binaire
|	a | b	OU binaire
&&	a && b	ET logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient fausse
||	a || b	OU logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient vraie
? :	a ? b : c	opérateur conditionnel : renvoie la valeur b ou c selon l'évaluation de l'expression a (si a alors b sinon c) : b et c doivent retourner le même type

LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

II - LES OPERATEURS

5. Opération sur les chaines de caractères

OPERATEUR	SIGNIFICATION
+	concaténation

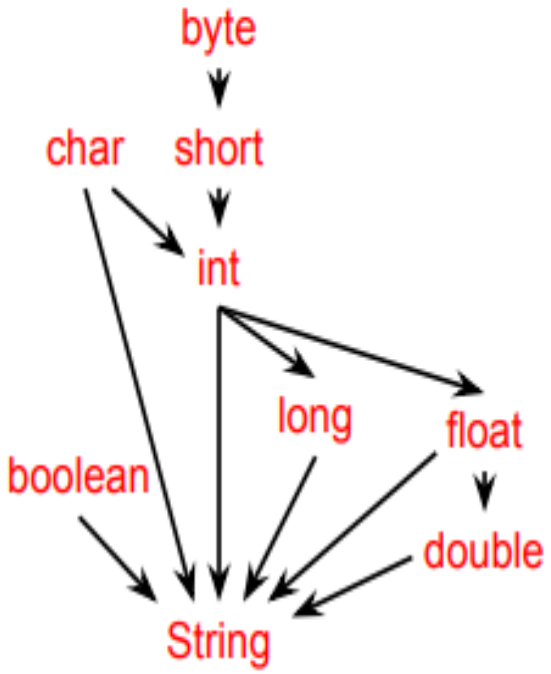
```
String s1 = "Bon";
 String s2 = "jour";
String s3 = s1 + s2 ;  //Après ces instructions s3 vaut "Bonjour"
```

LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

III - CONVERSION AUTOMATIQUE DE TYPE

Avant d'effectuer une opération, les valeurs peuvent être converties vers un type plus fort



Type le plus faible

Type le plus fort

EXEMPLE

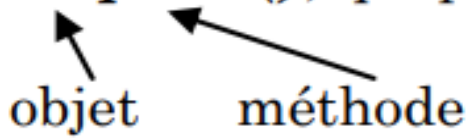
```
int x = 3;
Double y = x + 2.2; /* 5.2
*/
String s1 = "Coucou" + x;
    /* Coucou3 */
char c = 'a';
String s2 = "Coucou" + c;
    /* Coucoua */
```

LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

IV - LECTURE ET AFFICHAGE DE DONNÉES

Dans le langage java, les opérations de sortie sont donc réalisées grâce à l'instruction : ***System.out.print()***, qui permet d'afficher des informations à l'écran.



Par exemple,

l'instruction : **System.out.print("Bonjour ") ;** affiche à l'écran le texte:

Bonjour

int A=10, B=5 ;

System.out.print("La somme de " + A + "et " + B + " = "+ (A+B)) ; affiche à l'écran le texte:

La somme de 10**et**5 **= 15**

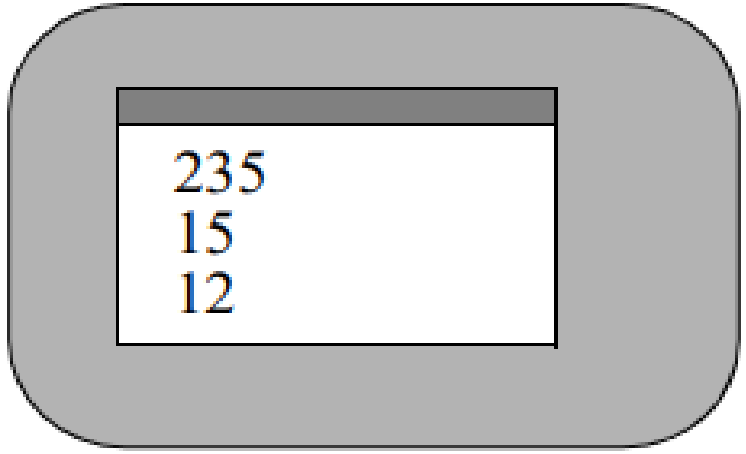
LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

IV - LECTURE ET AFFICHAGE DE DONNÉES

System.out.println (): affiche simplement le résultat
System.out.print () : provoque un passage au début de la ligne suivante après l'affichage du résultat

```
System.out.println(235);
System.out.println(9+6);
System.out.println(12);
```



LANGAGE JAVA

Chapitre 2: ELEMENTS DE BASE

IV - LECTURE ET AFFICHAGE DE DONNÉES

Les opérations d'entrée sont réalisées par l'intermédiaire de la classe **Scanner** grâce aux instructions suivantes :

Scanner lire=new Scanner(System.in) ;

int i=lire.nextInt();

Les méthodes de lecture ont pour nom d'appel :

- nextByte() pour saisir une valeur de type byte ;
- nextShort() pour saisir une valeur de type short ;
- nextInt() pour saisir une valeur de type int ;
- nextLong() pour saisir une valeur de type long ;
- nextFloat() pour saisir une valeur de type float ;
- nextdouble() pour saisir une valeur de type double ;
- next() pour saisir une valeur (un mot, sans l'espace blanc) de type String ;
- nextLine() pour saisir une phrase de type String ;
- next().charAT(0) pour saisir une valeur de type char ;

LANGAGE JAVA

Chapitre 3: DECLARATION

Les types spécifient la nature des données manipulées par les programmes, constitués d'"instructions" qui agissent sur des "données"
Un programme simple possède la structure suivante:

```
class nom du programme {

    déclarations de données globales

    définitions de fonctions

    procédure principale
}
```

LANGAGE JAVA

Chapitre 3: DECLARATION

Les déclarations de **données globales** permettent de définir des données qui seront utilisables depuis tout le texte du programme, par opposition aux **paramètres** et **données locales** des diverses fonctions du programme qui ne sont utilisables que depuis le texte de ces fonctions.

I- DÉCLARATIONS DE DONNÉES

D'une façon très générale une donnée est quelque chose qui :

- joue un certain rôle,
- a un nom (on dit aussi un identificateur),
- possède une valeur d'un certain type.

LANGAGE JAVA

Chapitre 3: DECLARATION

I- DÉCLARATIONS DE DONNÉES

Une donnée peut être **globale** ou **locale** :

- Une **donnée globale** est utilisable depuis toutes les fonctions de la classe qui constitue le programme. Une déclaration de donnée globale apparaît au premier niveau de la classe, en dehors de toute fonction. De plus, une donnée globale peut être statique ou non statique. Une donnée statique est créée dès le début d'exécution du programme. Dans le cas de programmes simples, les données globales seront toujours statiques. Elles sont précédées du mot réservé **static**
- Une **donnée locale** n'est utilisable que depuis le texte de la fonction où elle est déclarée.

LANGAGE JAVA

Chapitre 3: DECLARATION

I- DÉCLARATIONS DE DONNÉES

Une donnée peut être **une variable** ou **une constante** :

- La valeur associée à une variable peut être modifiée en cours d'exécution, au moyen d'une instruction d'affectation.
- La valeur associée à une constante est fixée une fois pour toute à l'endroit de sa déclaration. Une déclaration de constante est précédée du mot réservé **final.**

LANGAGE JAVA

Chapitre 3: DECLARATION

I- DÉCLARATIONS DE DONNÉES

Forme générale de déclarations de données dans un programme

```
class leProgramme {
   ...
   static final type nom = valeur ; ← constante globale
   static type nom ; ← variable globale
   ...

   static void ppp(...) {← définition de fonction
      final type nom = valeur ; ← constante locale à la procédure ppp
      type nom; ← variable locale à la fonction ppp
      ...
   }

   public static void main(String[] z) { ← procédure principale
      final type nom = valeur ; ← constante locale à la procédure principale
      ...
   }
}
```

LANGAGE JAVA

Chapitre 3: DECLARATION

I- DÉCLARATIONS DE DONNÉES

Intérêt des déclarations de constantes

- Une plus **grande lisibilité** du programme, qui entraine une bonne compréhension par les utilisateurs: c'est-à-dire permettre des modifications dues aux changements des besoins des utilisateurs du programme. Si la constante doit être modifiée, dans une version ultérieure, seule la déclaration de constante est à modifier.
- **Nommer** des résultats intermédiaires et ainsi éviter des expressions trop grosses qui pourraient nuire à la lisibilité

LANGAGE JAVA

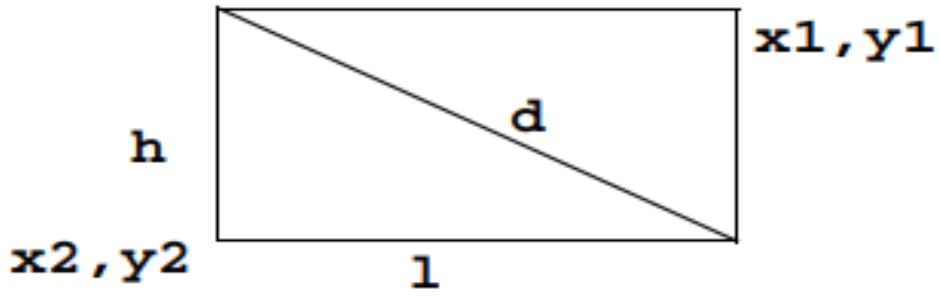
Chapitre 3: DECLARATION

I- DÉCLARATIONS DE DONNÉES

Intérêt des déclarations de constantes

soit un rectangle défini par les coordonnées de son coin supérieur droit (**x1,y1**) et de son coin inférieur gauche (**x2,y2**). On peut calculer la dimension de sa diagonale par :

```
final double d = Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
```



mais on préférera peut-être utiliser les notion intermédiaires de ***hauteur*** et de ***largeur*** :

```
final double h = x1-x2;
final double l = y1-y2;
final double d = Math.sqrt(h*h+l*l);
```

LANGAGE JAVA

Chapitre 3: DECLARATION

I- DÉCLARATIONS DE DONNÉES

Intérêt des déclarations de constantes

- calculer une seule fois une valeur qui doit être utilisée à plusieurs reprises soit par exemple le calcul de la surface de base et du volume d'un parallélépipède. Le volume est lui-même obtenu en multipliant la surface de base par la hauteur. On veut calculer une fois la surface de base :

```
class Parallelepipede {
  public static void main(String[] arg) {
    int largeur=...; int longueur=...; int hauteur=...;

    final int surfaceDeBase=largeur*longueur;

    System.out.print("surface de base = ");
    System.out.println(surfaceDeBase);
    System.out.print("volume = ");
    System.out.println(surfaceDeBase*hauteur);
  }
}
```

LANGAGE JAVA

Chapitre 3: DECLARATION

II- DÉFINITION DE FONCTIONS

Une fonction est une “collection identifiée d’instructions qui fait quelque chose”. Une fonction est destinée à être appelée

Une définition de fonction a la forme suivante :

```
static typeDuRésultat nomDeLaFonction (type1 param1, type2 param2...) {
        déclarations locales et instructions
}
```

Le type du résultat est indiqué devant le nom de la fonction, le type et le nom de chaque paramètre sont indiqués entre parenthèses à la suite du nom de la fonction. Ces paramètres sont appelés paramètres formels

Si une fonction n’a pas de paramètre, il faut tout de même mettre les parenthèses :

```
static typeDuRésultat nomDeLaFonction () {...}
```

LANGAGE JAVA

Chapitre 3: DECLARATION

II- DÉFINITION DE FONCTIONS

il existe des fonctions qui ne rendent aucun résultat. On a l'habitude d'appeler procédure une telle fonction. Dans ce cas on utilise le mot réservé **void** à la place du **type du résultat** :

```
static void nomDeLaProcédure (type1 param1, type2 param2...) {
        déclarations locales et instructions
}
```

Pour rendre son résultat, une fonction dispose de l'instruction de retour dont la forme générale est :

return expression;

EXEMPLE

```
static int moyenneDe3Nombres(int i, int j, int k) {
    return (i+j+k)/3;}
}
```

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

classification des types offerts par la plupart des langages de programmation :

<table>
<tr><td rowspan="7">types</td><td rowspan="5">types primitifs</td><td rowspan="4">scalaires</td><td>entiers</td><td>12</td></tr>
<tr><td>réels</td><td>12.0</td></tr>
<tr><td>caractères</td><td>'w'</td></tr>
<tr><td>booléens</td><td>false</td></tr>
<tr><td rowspan="2">composés</td><td>tableaux</td><td>0 12
1 2
2 25
3 12</td></tr>
<tr><td>chaînes</td><td>"coucou"</td></tr>
<tr><td rowspan="3">types programmés</td><td>énumérés</td><td colspan="2">Couleur = {bleu,blanc,rouge}</td></tr>
<tr><td>structures</td><td colspan="2">Personne = <nom,age> ex. <toto,19></td></tr>
<tr><td>classe</td><td colspan="2">class Voiture {
...
void accelerer() {...}
void freiner() {...}
}</td></tr>
</table>

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

1. Types primitifs scalaires

En Java offre les types primitifs scalaires suivants :

domaine représenté	*types Java*	*notations de valeurs*
nombres entiers	`int, long, short, byte`	`12 -4567`
nombres réels	`double, float`	`3.141592 6.02E23`
caractères	`char`	`'a' 'z' '?' '\n'`
valeurs logiques	`boolean`	`true false`

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

Un **tableau** de n entiers correspond à peu près à l'idée mathématique d'un vecteur de n composantes entières.
Une **chaîne de caractères** est une suite finie de caractères, souvent utilisée pour constituer des textes lisibles par un être humain.

Tableaux à une et deux dimensions

Un tableau est un ensemble indexé de données d'un même type

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

Tableaux à une et deux dimensions

- Création d'un tableau: on définit un type comme pour une variable de base, auquel on ajoute des crochets ouvrant et fermant [ ] pour indiquer qu'il s'agit d'un tableau. Sa taille est donnée à son instanciation.

```
int tableau[] = new int[50]; // déclaration et allocation

//OU
int[] tableau = new int[50];

//OU

int tab[]; // déclaration
tab = new int[50]; //allocation
```

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

Tableaux à une et deux dimensions

La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création.

```
int tableau[] = {10,20,30,40,50};
```

Les tableaux existent en 1 et 2 dimensions. Les tableaux à 2 dimensions sont considérés comme des tableaux de tableaux.

```
float tableau[][] = new float[10][10]; //Tableau à 2 dimensions

int tableau[3][2] = {{5,1},{6,2},{7,3}}; //Tableau à 2 dimensions
```

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

Tableaux à une et deux dimensions

- Parcours d'un tableau se fait à l'aide de la boucle **for**. On utilise la syntaxe de type **for each** si on n'a pas besoin de connaître l'indice de chaque élément à traiter

```
int[] monTableau = new int[ 10 ];
for ( int i = 0; i < monTableau.length; i++ ){
      monTableau[i] = i + 1;
}
```

La variable **length** retourne le nombre d'éléments du tableau.
monTableau[i] désigne le contenu du tableau à l'indice i.

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

C'est une suite finie de caractères, souvent utilisée pour constituer des textes lisibles par un être humain. On utilise une classe particulière, nommée **String**, fournie dans le **package java.lang**.

Les variables de type String ont les caractéristiques suivantes :

- leur valeur ne peut pas être modifiée
- on peut utiliser l'opérateur + pour concaténer deux chaînes de caractères

en JAVA un string est un **objet**

La classe **String**, est utilisée pour les chaînes fixes (non modifiables),

La classe **StringBuffer** pour les chaînes variables (modifiables).

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

new String("Bonjour")	création d'une chaîne initialisée
new String(String str);	création d'une chaîne initialisée
new String();	création d'une chaîne non initialisée
new String(int taille);	création d'une chaîne non initialisée
boolean equals(String)	comparaison de chaînes ***if (str1.equals(str2))...***
int compareTo(String)	Cette méthode retourne 0 si les deux chaines sont égales, une valeur négative si str1 est plus petit que str2, ou une valeur positive si str1 est plus grand que str2. ***str1.compareTo(str2)***
int indexOf(char)	recherche d'un caractère et obtention de son rang

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

int indexOf(char, int)	recherche d'un caractère et obtention de son rang en faisant démarrer la recherche à partir d'un rang donné en second paramètre
int indexOf(String)	recherche d'une sous chaîne et obtention de son rang
int indexOf(String, int)	recherche d'une sous chaîne et obtention de son rang en faisant démarrer la recherche à partir d'un rang donné en second paramètre
char charAt(int)	extraction d'un caractère par son rang
String subString(int,int)	extraction d'une sous chaîne par rangs de début et de fin
int length()	Longueur de la chaine
String concat(String)	concaténation : on peut aussi utiliser l'opérateur +

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

Un objet de type **StringBuffer** représente une chaîne de caractères modifiable. On peut changer un caractère, ajouter des caractères au tampon interne de l'objet défini avec une capacité qui peut être changée si besoin est.

Un StringBuffer évolue dynamiquement. Si on augmente la longueur de son contenu, l'espace mémoire alloué est automatiquement augmenté. Plusieurs méthodes importantes différencient la classe StringBuffer et la classe String, dont : length, capacity, setLength, charAt, setCharAt, append, insert et toString

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

new StringBuffer("Bonjour")	création d'une chaîne initialisée
new StringBuffer(String str);	création d'une chaîne initialisée
new StringBuffer();	création d'une chaîne non initialisée construit un tampon ayant 0 caractères mais d'une capacité par défaut de 16 caractères
new StringBuffer(int taille);	création d'une chaîne non initialisée construit un tampon de capacité n caractères
StringBuffer nom = StringBuffer.ValueOf(variable)	création d'une chaîne à partir d'une variable primitive

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

Les méthodes ci-face concatènent au tampon de l'objet courant la représentation sous forme de caractères de leur argument (boolean, int, etc...)

StringBufffer append (boolean)
StringBufffer append (char)
StringBufffer append (char[ ])
StringBufffer append (double)
StringBufffer append (float)
StringBufffer append (int)
StringBufffer append (long)
StringBufffer append (Object) :
StringBufffer append (String)

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

Les méthodes ci-face insèrent dans le tampon de l'objet, à partir de l'indice du paramètre 1, la représentation sous forme de caractères de leur argument.

StringBufffer insert (int, boolean)
StringBufffer insert (int, char)
StringBufffer insert (int, char[ ])
StringBufffer insert (int, double)
StringBufffer insert (int, float)
StringBufffer insert (int, int)
StringBufffer insert (int, long)
StringBufffer insert (int, Object)
StringBufffer insert (int, String)

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

I- TYPES PRIMITIFS

2. Types primitifs composés

chaînes de caractères

D'autres méthodes fournissent la capacité de l'objet de type StringBuffer, l'accès à un caractère à partir de son indice, la longueur de la chaine, la chaine inverse, ou l'objet de type String équivalent.

int capacity ( )	fournit la capacité de cet objet (nombre maximum de caractères)
char charAt (int n)	fournit le caractère d'indice n
StringBuffer reverse ()	inverse les caractères de l'objet (gauche-droite devient droite-gauche)
String toString ()	convertit en un objet String
ensureCapacity (int n)	la capacité de l'objet est au moins n sinon un nouveau tampon est alloué
void setCharAt (int index, char ch)	le caractère index reçoit ch (0<=index<longueur)

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

II- TYPES PROGRAMMES

1. Types programmés énumérés

Une énumération est un type de données particulier, dans lequel une variable ne peut prendre qu'un nombre restreint de valeurs. Ces valeurs sont des constantes nommées, par exemple MADAME, MADEMOISELLE, MONSIEUR.

Déclaration d'une énumération: La façon la plus simple de déclarer une énumération consiste à l'écrire lorsque l'on crée une variable, comme dans le code qui suit.

```
enum Civilite {MADAME, MADEMOISELLE, MONSIEUR} ;
    Civilite civilite = Civilite.MADAME ;
```

LANGAGE JAVA

Chapitre 4: MANIPULATION DES TYPES DE DONNEES

II- TYPES PROGRAMMES

1. Types programmés énumérés

Déclaration d'une énumération: Il est possible de déclarer une énumération dans un fichier séparé, comme une classe, en remplaçant simplement le mot-clé class par le mot-clé **enum**

Une énumération est en fait une classe, d'où cette appellation de classe énumération. Cette classe étend la classe **Enum**, qui elle-même étend la classe Object, comme toutes les classes en Java.

```
public enum Civilite {  // dans le fichier Civilite.java
    MADAME, MADEMOISELLE, MONSIEUR
}
```