Cours de Java

Ahmed Zidna, Bureau : B37.
Département Informatique de l'IUT
Univerisité de Lorraine, Ile du Saulcy, F-57045 METZ
ahmed.zidna@univ-lorraine.fr

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Variables I

- En Java, toutes les variables sont déclarées et typées. Une variable est une zone mémoire pour stocker une donnée. Elle est définie par :
  - son adresse : celle du premier octet qu'elle occupe
  - son type : il définit sa taille
  - sa valeur : c'est son état à un instant donné.

```
1    String ch; float x; int [] Tab; char c='a';
```

- Toute variable est stockée à une adresse mémoire et occupe un nombre d'octet en fonction du type.
- Pour comparer : >, >=, <, <=, ==, ! =

Les types I

On distingue deux sortes de types :

- les types primitifs = booléen, caractère, entier, réel.
- les autres types : objets définis à partir d'une classe avec new, les tableaux, les chaînes

```
1   boolean trouve=false;
    float  y =7.5;
3   String ch;
    ch=new String("Bonjour");
5   int  [] tab =new int[10];
```

- En Java il n'est pas possible, d'obtenir une référence sur une variable de type int déclarée comme telle.

Types primitifs I

1. le type booléen :
   - **boolean** : prend les valeurs **false** et **true**.
2. les caractères
   - **char** : au format Unicode et est codé sur 16 bits.
3. Les entiers :
   - **byte** : valeur codée sur 8 bits, de -2^7 à $2^7 - 1$
   - **short** : valeur codée sur 16 bits, de -2^{15} à $2^{15} - 1$
   - **int** :valeur codée sur 32 bits, de -2^{31} à $2^{31} - 1$
   - **long** : valeur codée sur 64 bits, de -2^{63} à $2^{63} - 1$
4. Les réels :
   - **float** : sur 32 bits conforme à la norme *IEEE*754
   - **double** sur 64 bits conforme à la même norme

Types primitifs I

- Les conversions entre type numérique se font de façon implicite dans ce sens :

  byte → *short* → *int* → *long* → *float* → *double*

- Pour les conversions explicites : utiliser l'opérateur de **cast** :

```
1  int u; long v; float x; double y=3E8;
   char c1,c2='B';
3  u=(int)v;
   x=(float)y;
5  c1=(char)u;
   u=(int)c2;
```

- les constantes sont définies comme suit :

```
   final double PI=3.14;
2  final int x=5;
```

Types non primitifs I

1. les Tableaux :

```
double [] Tab={2.5,5.78, 3.14};
int [] T=new int[10];
```

2. Les chaînes de caractères : String

```
String chaine1="toto";
String chaine2= new String("Bonjour");
```

3. Les objets instances d'une classe :

```
Personne p1=new Personne("Durand",45);
Personne p2=new Personne("Dupont",35);
```

Classes enveloppes I

Il existe des types qui enveloppent les types primitifs (wrappers).

- Chaque type primitif possède une classe associée : Byte, Short, Integer, Float, Double, Character
- Chaque type est muni de nombreuses fonctionnalités :
- Exemple 1 : conversion d'une chaîne à un nombre entier

```
int n;
String chaine1="123";
int n=Integer.parseInt(chaine1);
System.out.println("valeurden="+n);//n=123
```

- Exemple 2 : conversion d'une chaîne à un nombre entier

```
int a=Integer.parseInt("110", 2);//a=6
int b=Integer.parseInt("123"); //b=123
double c=Double.parseDouble("5");//c=5.0
int d=Integer.parseInt("44",16); //d=68
```

Structure de contrôle I

- Instruction if...else

```
if (condition)
{instruction ou bloc}
else
{instruction ou bloc}
```

- Exemple

```
public static void main(String [] args){
int max, a=5, b=3;
if (a>b)
{max=a; System.out.println(a+"estplusgrand");}
else
{max=b; System.out.println(b+"estplusgrand");}
}
```

Structure de contrôle I

- instruction de répétition while

```
1  while(condition){
   instruction ou bloc
3  }
```

- Il n y a aucune exécution si la condition est initialement fausse.
- Exemple : Calcul de la somme des nombres impairs.

```
1  public static void main(String [] args){
   int i =1, somme=0,dernier;
3  System.out.println ("saisirledernier");
   Scanner sc=new Scanner(System.in);
5  dernier=sc.nextInt();
   while(i<dernier){
7  somme +=i;
   i=i+2;
9  }
   System.out.println("lasommeest:"+ somme);
11 }
```

structure de contrôle I

- instruction do...while

```
1 do{
  instruction ou bloc
3 }while(condition);
```

- La boucle est exécutée tant que la condition est vraie et une fois au moins.

- Exemple

```
1 public static void main(String [] args){
  int note;
3 do{
  System.out.println("Entrerunenote");
5 Scanner sc=new Scanner(System.in);
  note=sc.nextInt();
7 }while((note<0)||(note>20));
  }
```

Structure de contrôle I

- instruction for

```
for(initialisation;condition;mise a jour)
{
 instruction ou bloc
}
```

- Exemple

```
for (int i=0;i <10;i++)
System.out.println("lavaleurdei est"+i);

for (int i=10; i>0 ;i--)
System.out.println("lavaleurdei est"+i);

for(int i=0;i<n;i++) tab[i]=0;

for (int i=0,int j=n;i<j;i++,j--)
```

Structure de contrôle I

- instruction switch...case

```
1  // expression est de type char, byte, short, int ou long
   switch(expression){
3  case const1 : instruction1; break;
   case const2 : instruction2; break;
5   .....
   default : instruction; break;}
```

- **Exemple**

```
   enum Couleur{ROUGE,NOIR,BLANC,ORANGE,VERT}
2  public static main{String [] args){
   Couleur c=Couleur.BLANC;
4  switch(c){
   case VERT :System.out.println("jepasse");break;
6  case ORANGE:System.out.println("jefreine");break;
   case ROUGE :System.out.println("jem'arrete");break;
8  default:System.out.println("jebronze");break;
   }}
```

Fonctions classiques I

1. Soit membre d'une classe :

   $\Rightarrow$ est accessible à travers un objet :

```
1   String ch=new String("Bonjour");
    ch.length(); ch.charAt(3);
```

2. Soit non membre d'une classe :

   $\Rightarrow$ déclarée avec le mot clé **public static**.

   $\Rightarrow$ accessible par le nom de la classe.

```
    String chaine2 = String.valueOf(123);
2   String chaine3 = String.valueOf(12.5);
```

Fonctions classiques I

- fonction main

```
public class AutreProgramme{
public static void main(String [] arg){
int a = 4; b = 5;
c = max(a,b);
System.out.println("leplusgrand"+a+"et"+b+"est"+c);
}
```

- Fonction max

```
public static int max( int x, int y){
int m;
if (x>=y) m=x; else m=y;
return m;
}
}
```

Modularité I

- On peut développer une bibliothèque de méthodes afin d'être utilisée dans d'autres programmes

```
public class Operation{
// Fonction somme
public static int somme(int a, int b){
int res=a+b; return res;
}
// Fonction Produit
public static int produit(int a, int b){
int res=a*b; return res;
}
// Fonction diffrence
public static int diffrence(int a, int b){
int res=a-b; return res;
}
// Fonction afficher
public static void afficher(int res)
{ System.out.println("laresultat="+res);
}
```

- Voici un programme pour tester la bibliothèque **Operation**

```
1  public class TestOperation{
   public static void main(String[] args){
3  int x = 2;
   int y = 3;
5  int s = Operation.somme(x,y);
   Operation.afficher(s);
7  int p = Operation.produit(x,y);
   Operation.afficher(p);
9  int d = Operation.diffrence(x,y);
   Operation.afficher(d);
11 }
   }
```

Fonctions et passage des paramètres I

1. Dans une fonction, un paramètre de type primitif est passée par valeur

```
    public static void main(String [] arg){
2   int a=2;
    incremente(a);
4   System.out.println("leresultatesta="+a);
    }
```

```
1   public static void incremente(int x){
    x=x+1;
3   }
```

2. Une variable de type non-primitif est passée par référence.

Fonctions classiques I

- Pour incrementer l'entier x, on utilise un objet qui enveloppe un entier

```
1  class MonEntier{
   int val;
3  MonEntier(int a){val =a;}
   }
```

```
   public static void main(String [] arg){
2  MonEntier X=new MonEntier(2);
   incremente(X);
4  System.out.println("leresultatestX="+X);
   }
```

- Voici la fonction incremente

```
1  public static void incremente(MonEntier A){
   A.val=A.val+1;
3  }
```

Chaînes de caractères : String I

- Les chaînes de caractères de type String, sont des constantes, elles ne peuvent pas être modifiées.

```
1  String chaine = "Bonjour";
   System.out.println(chaine);
```

- Les chaînes de caractères peuvent être concaténées.

```
   String chaine1="Bonjour";
2  String chaine2="toutlemonde!";
   chaine3=chaine1+chaine2;
4  System.out.println(chaine3);
```

Chaînes de caractères : String I

- Plusieurs fonctions pour manipuler le type String

```
ch.length(), ch.toString(),ch.indexOf('u'),
ch.substring(0,4), ch.startsWith("www"), ch.At(2),
ch.toUpperCase(), ch.isEmpty()
```

```
String chaine1="Bonjour";
int i= chaine1.length(); //i=7
String chainee2=chaine1.substring(0,2);
Boolean test;
test=chaine1.equals(chaine2); //test=false
```

```
String ch="abcdef" ;
char [] tabCar ;
tabCar=ch.toCharArray( ) ;
```

Chaînes de caractères : StringBuffer I

- Les StringBuffer sont des chaînes qui peuvent être modifiées en conservant leurs adresses initiales.

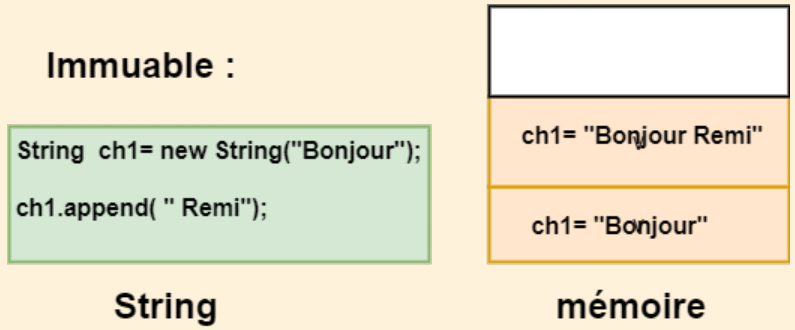
```
1  StringBuffer chaine;
   chaine=new StringBuffer("Bon");
3  chaine.append("jour"); // ajout a la fin de chaine
   System.out.println(chaine); // ecriture de Bonjour
```

- Pour comparer : ==, equals et compareTo

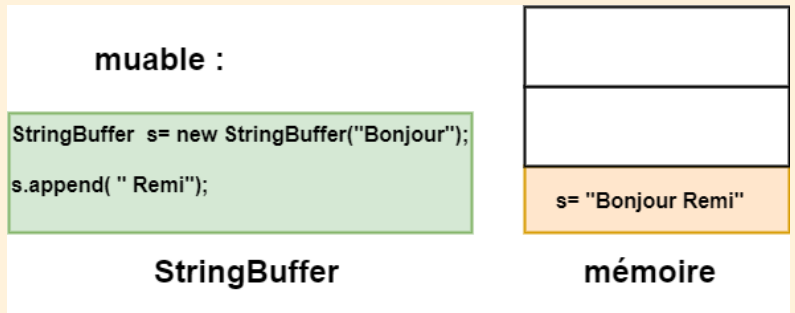
```
   StringBuffer ch1= new StringBuffer("Bonsoir");
2  StringBuffer ch2= new StringBuffer("Bonsoir");
   boolean val=(ch1==ch2); -----> FALSE
4  boolean val2= ch1.equals(ch2);----->TRUE
   int val3= ch1.compareTo(ch2); ----->0
```

différence entre String et StringBuffer I

- Les chaines String sont immutables : ne peuvent pas être modifiées.



- Les chaines StringBuffer sont mutables :peuvent être modifiées en gardant leur adresse initiale.



StringBuilder I

- Les classes StringBuffer et StringBuilder

  $\Rightarrow$ produisent des objets qui peuvent changer de valeur.

  $\Rightarrow$ Ces deux classes ont exactement les mêmes méthodes.

- Les méthodes de la classe StringBuffer sont synchronisés,

  $\Rightarrow$ lorsque plusieurs thread sont utilisés sur une même chaine.

  $\Rightarrow$un unique thread à accès à la chaine à l'instant t : "thread safe".

- Les méthodes de la classe StringBuilder ne sont pas synchronisées,

  $\Rightarrow$ elles sont donc plus rapides que celles de la classe StringBuffer, mais ne sont pas "thread safe".

Tableaux I

- Les tableaux sont des objets, il faut les créer par l'opérateur new.

```
1  int [] tab; //declaration de tab
   tab = new int[10]; // creation de tab
```

- ou bien

```
   double[] T1={5.3, 4.5, 3.1}; // initialisation
2  int [] T2= new int[10]; // declaration et creation
```

- ou encore :

```
   static final int DIM_TAB=10;
2  int [] tab = new int[DIM_TAB];
   for(int i=0; i< tab.length; i=i+1)
4  tab[i]=5*i;
```

Tableaux I

- Une exception de type IndexOutOfBoundsException est levée quand l'indice est erroné.

```
try{
j=tab[tab.length];
}catch (IndexOutOfBoundsException e){
System.out.println("Indiceincorrect"+e);
}
```

- Tableaux multidimensionnels :

```
final int DIM1=4;
final int DIM2=3;
double [][] mat= new double[DIM1][DIM2];
for(int i=0;i<DIM1;i=i+1)
    for (int j=0;j<DIM2;j=j+1)
        mat[i][j]=i*j;
```

Tableaux I

- Exercice : On se propose d'écrire une méthode qui crée et renvoie l'adresse d'un triangle de Pascal dont la dimension est passée en paramétre. Un triangle de Pascal est la matrice triangulaire inférieure P des coefficients binômiaux. Les coefficients binômiaux se calculent à l'aide des relations.
  $\binom{n}{n} = \binom{n}{0} = 1$
  $\binom{n}{p} = \binom{n-1}{p} + \binom{n-1}{p-1}$

Tableaux II

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

TABLE – triangle de Pascal.

Tableaux I

- Voici la fonction Pascal qui construit le triangle de Pascal

```
static int [][] Pascal(int n){
int [][] P=new int[n+1][];// n+1 lignes
for(int i=0;i<=n;i++) {//calcul de la ligne i
P[i]=new int[i+1];//creation de la ligne i
P[i][0]=P[i][i]= 1;// initialisation des extremites
for(int j=1;j<i;j++) //calcul des autres coefficients
P[i][j]=P[i-1][j]+P[i-1][j-1];
}
return P;
}
```

- Voici une fonction afficher qui affiche le tableau de Pascal

```
static void afficher(int[][]T){
for(int i=0; i<T.length; i++){
for(int j=0; j<T[i].length; j++)
System.out.print( T[i][j]+"");
System.out.println();
}
}
```

Tableaux II

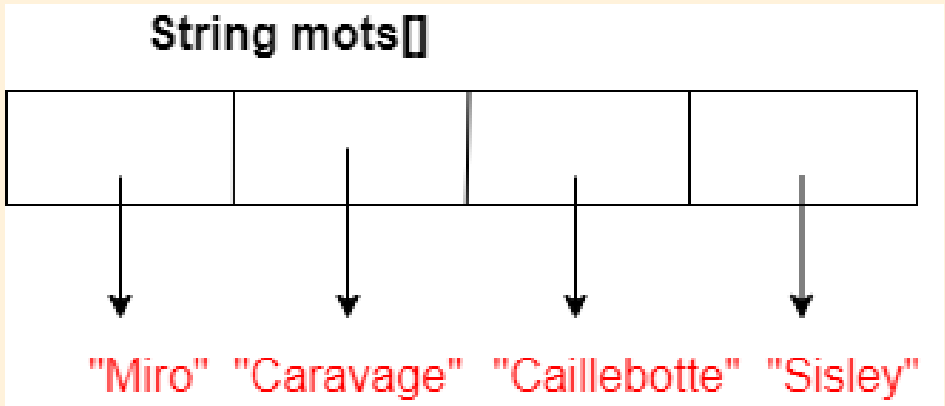
- Voici la fonction main pour tester

```
1  public static void main(String[] args){
   afficher(Pascal(6)));
3  }
```

Tableaux I

- Les éléments des tableaux de chaînes de caractères ou d'objets ne sont pas les chaînes ou les objets eux-mêmes, mais leurs adresses.

```
1   String mots[]={"Miro", "Caravage", "caillebotte" ,  "Sisley" };
```




```
1   for (int i=0;i<mots.length; i++)
     System.out.println(mots[i].toUpperCase());
```