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 $\rightarrow$ *short* $\rightarrow$ *int* $\rightarrow$ *long* $\rightarrow$ *float* $\rightarrow$ *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 ch="toto";
```

3 Les objets définis à partir d'une classe :

```
class Personne{
String nom; int age;
Personne(String s, int a){
nom=s;
age=a;
}}
Personne p=new Personne("Durand" ,45);
```

Classes enveloppes I

- Il existe des types encapsulant les types primitifs (wrappers).
- Chacun des types primitifs possède une classe associée : Byte, Short, Integer, Float, Double, Char
- La classe Integer permet la définition de valeurs entières comme objets.
- Chaque type est muni de nombreuses fonctionnalités : convertir une chaîne de caractères en type simple.
- Exemple : conversion de chaîne en nombre entier

```
1  int n;
   String s = "123";
3  Integer i;
   i=new Integer(s);//transforme la chaine en Integer
5  n=i.intValue();//transforme un Integer en int
```

Structure de contrôle I

- Instruction if...else

```
1     if (condition)
        {instruction ou bloc
3       }
        else
5       {instruction ou bloc
        }
```

- **Exemple**

```
  public static void main(String [] args){
2 int max, a=5, b=3;
  if (a>b){max=a;
4         System.out.println(a+"est␣le␣plus␣grand");
        }
6 else { max=b;
       System.out.println(b+"est␣plus␣grand");
8       }
  }
```

Structure de contrôle I

- instruction de répétition while

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

- Il n y a aucune exécution si la condition est initialement fausse.
- **Exemple :** Voici un programme qui calcule la somme des nombres impairs jusqu'au nombre entré au clavier.

Structure de contrôle II

```
   public static void main(String [] args){
2  int i =1,somme=0,dernier;
   Scanner sc=new Scanner(System.in);
4  System.out.println ("saisir le dernier nombre");
   dernier=sc.nextInt();
6  while(i <dernier)
     { somme +=i;
8      i=i+2;
     }
10 System.out.println("la somme est :"+ somme);
   }
```

- 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 i=0;
3   do{
    System.out.println("la␣valeur␣de␣l'indice␣est␣:" +i);
5   i=i+1;
    }while(i<5);
7   }
```

Structure de contrôle I

- instruction for

```
1   for(initialisation;condition;mise à jour)
      {
3       instruction ou bloc
      }
```

- **Exemple**

```
    for (i=0;i <10;i++)
2   System.out.println("la valeur de i est"+i);

4   for (i=10; i>0 ;i--)
    System.out.println("la valeur de i est"+i);
6
    for(i=0;i<n;i++) tab[i]=0;
8
    for (i=0,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;
7  }
```

- **Exemple**

```
1  public static main{String [] args){
   public enum Couleur{ROUGE,NOIR,BLANC,ORANGE,VERT;}
3  Couleur c;
   switch(c){
5  VERT :System.out.println("je␣passe");break;
   ORANGE:System.out.println("je␣freine");break;
7  ROUGE :System.out.println("je␣m'arrete");break;
   default:System.out.println("je␣bronze");break;}}
```

Fonctions classiques I

Une fonction en Java est

- Soit membre d'une classe.
- Soit non membre ou fonction ordinaire :
- Une fonction non membre est déclarée avec le mot clé **public static**.
- Dans une fonction, les variables de type primitif est passée par valeur
- Une variable de type non-primitif est passée par référence.

Fonctions classiques I

```
public class AutreProgramme{

public static void main(String [] arg){
int a = 4; b = 5; c = max(a,b);
System.out.println("le plus grand "+a+"et"+b+"est"+c+"\n");
}

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

Fonctions classiques I

Les paramétres sont passés par valeur

```
1  public static void main(String [] arg){
   int a=2;
3  incremente(a);
   System.out.println("le résultat est a="+a);
5  }
```

- Voici la fonction incremente

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

Fonctions classiques I

Pour modifier x, on utilise une classe enveloppe

- Exemple

```
1  class MonEntier{
   int val;
3  }
```

```
1  public static void main(String [] arg){
   MonEntier x=new MonEntier();
3  x.val=2;
   incremente(x);
5  System.out.println("le résultat est x="+x);
   }
```

- Voici la fonction incremente

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

Fonctions classiques I

```
1  public static void echange(int a, int b){
   int aux;
3  aux=a; a=b; b=aux;
   }
```

```
   public static void main(String [] arg){
2  int x=5, y=3;
   echange(x,y);
4  System.out.println("le␣résultat␣de␣l'echane␣est␣x="+x+ "␣y=
       "+y);
   }
```

- Les valeurs de x et de y n'ont pas changé car les paramétres sont passés par valeur.

Fonctions classiques I

Pour echanger a et b, on utilise une classe enveloppe

```
1  class MonEntier{
   int val;
3  }
```

- Voici la classe **TestMonEntier**

```
1  public class TestMonEntier {
   public static void main(String [] args){
3  MonEntier a= new MonEntier();
   MonEntier b= new MonEntier();
5  a.val=5; b.val=3;
   echange(a,b);
7  System.out.println("␣a␣est␣=␣"+a.val+ "␣b␣="+b.val);
   }
```

- Voici la fonction echange

Fonctions classiques II

```
   static void echange(MonEntier x, MonEntier y){
2  int aux= x.val;
   x.val= y.val;
4  y.val= aux;
   }
```

Modularité I

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

```
1  public class Operation{
   // Fonction somme
3  static int somme(int a, int b){
   int res;
5  res=a+b;
   return res;
7  }
   // Fonction Produit
9  static int produit(int a, int b){
   int res;
11 res=a*b;
   return res;
13 }
   }
```

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

```
public class TestOperation{
public static void main(String[] args){
int x = Integer.parseInt(args[0]);
int y = Integer.parseInt(args[1]);
int s = Operation.somme(x,y);
int p = Operation.produit(x,y);
System.out.println("Somme("+ x + "," + y+ ") = " + s);
System.out.println("Produit("+ x + "," + y+ ") = " + p);
}
}
```

```
java TestOperation 5 3
Somme(5,3)=8
Produit(5,3)=15
```

Chaînes de caractères : String I

- Toutes les chaînes de caractères, de type String, sont des constantes, elles ne peuvent pas être modifiées. Ce sont des objets non variables.
- Plusieurs fonctions pour manipuler le type String

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

```
1  String chaine; //declaration de chaine de caracteres
   chaine = "Bonjour"; // pas besoin de new,creation de la
       chaine
3  System.out.println(chaine);
```

- Les chaînes de caractères peuvent être concaténées.

Chaînes de caractères : String II

```
1  String chaine1="Bonjour";
   String chaine2="␣tout␣le␣monde␣!";
3  chaine3=chaine1+chaine2;
   System.out.println(chaine3); // ecriture de Bonjour tout le
        monde !
```

```
   String chaine1="Bonjour";
2  int i; i= chaine1.length(); // i vaut alors 7
   String chaine1="Bonjour";
4  String chaine2="Bon"; chaine3=chaine2+"jour";
   Boolean test;
6  test=chaine1.equals(chaine3); // test vaut true
```

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

Chaînes de caractères : String I

```
1  public class StringExemple {
   public static void main (String[] args) {
3  String phrase = "Changement␣inevitable";
   String ch1, ch2, ch3, ch4;
5  System.out.println ("Original␣:␣\"" + phrase + "\"");
   System.out.println ("Longueur:␣" + phrase.length());
7  ch1 = phrase.concat (",␣sauf␣pour␣ce␣qui␣marche.");
   ch2 = ch1.toUpperCase();
9  ch3 = ch2.replace ('E', 'X');
   ch4 = ch3.substring (3, 30);
11 System.out.println ("ch␣#1:␣" + ch1);
   System.out.println ("ch␣#2:␣" + ch2);
13 System.out.println ("ch␣#3:␣" + ch3);
   System.out.println ("ch␣#4:␣" + ch4);
15 System.out.println ("longueur␣:␣" + ch4.length());
   }
17 }
```

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"); // utiliser new
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
```

- Pour convertir un caractère en String :

```
1  char car='a';
   String s=String.valueOf(car);
```

Tableaux I

- Les tableaux sont des objets, il faut les créer par l'opérateur **new**
- Tout tableau T est indexé de 0 à $T.length - 1$.
- adresse de $T[i] = T + i$ (taille du type des éléments de T)

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

- ou bien

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

- ou encore :

```
1  static final int DIM_TAB=10;
   int [] tab = new int[DIM_TAB]; // bien preferable au
       precedent
3  for (int i=0; i< tab.length; i=i+1) tab[i]=5*i;
   // permet d'initialiser tous les elements du tableau
5  double[] tab_doub={5.3, 4.5, 3.1};
```

Tableaux II

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

```
try {
   j=tab_int[tab_int.length]; // leve
       IndexOutOfBoundsException
   }
catch (IndexOutOfBoundsException e) {
   System.out.println("Indice incorrect "+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;
//mat.length vaut DIM1, alors que mat[0].length vaut DIM2.
```

Tableaux III

- Exercice : On se propose d'écrire une méthode qui crée et renvoie l'adresse dŠun 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 IV

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$	$\binom{n}{7}$
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	
7	1	7	21	35	35	21	7	1

TABLE: triangle de Pascal.

Tableaux V

```
1  public class Testpascal{
   static int[][] Pascal (int n){..}
3  static void afficher(int[][]T){..}
   public static void main(String [ ] args){.. }
5  }
```

- Voici la fonction Pascal qui construit le triangle de Pascal

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

Tableaux VI

- Voici une fonction afficher qui affiche le tableau de Pascal

```
   static void afficher(int[][]T){
2  for(int i=0; i<T.length; i++){
   for(int j=0; j<T[i].length; j++)
4  System.out.print( T[i][j]+"␣");
   System.out.println();
6  }
   }
```

- Voici la fonction main pour tester

```
1  public static void main(String [ ] args){
   afficher(Pascal(Integer.parseInt(args[0])));
3  }
```

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

Tableaux VII

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

Tableaux I

- On veut inverser une suite de caractères dans un tableau par en faisant des permutations.
- Voici la classe **TestInverseTableau**

```
1  class TestInverseTableau{
   public static void main(String[ ] args){
3  char [ ] Tablecar ={'a','b','c','d','e','f'};
   int i,j ;
5  System.out.println("avant:"+ String.valueOf(Tablecar));
   for(i=0,j=5;i<j;i++,j--){
7  char c ;
   c = Tablecar[i];
9  Tablecar[i]= Tablecar[j];
   Tablecar[j]= c;
11 }
   System.out.println("apres : " + String.valueOf(Tablecar));
13 }}
```