Cours 3

- Tableaux
- Tableaux à 2 dimensions
- Tableaux d'objets
- Classes String et StringBuffer
- Enumération

Les **tableaux** ne sont pas ni des objets ni des types primitifs.

Un tableau se rapproche d'un objet

- Il est manipulé par référence (adresse)
- Il nécessite de *new* pour être défini

Déclaration

char tableau[] *ou* char[] tableau

Création

- En utilisant new

int tab[]; tab=new int[5];

char t2[]= new char[4];

- implicitement avec l'initialisation

int t2[]={1,2,3};

- La taille est fixée

- On peut accéder à la taille par le champ
  length

- Les indices entre 0 et length-1

- Dépassement d'indice : lancement d'exception *ArrayIndexOutOfBoundsException*

- Un objet ou un tableau qui n'est plus référencé est récupéré par ramasse-miettes (garbage)

```
public class TabAffec
{public static void main(String args[])
       {char t1[]={'b','o','n','j','o','u','r'};
       char t2[]={'h','e','l','l','o'};
       char t3[]={'x','x','x','x'};
       t3=t1;  t1=t2; t2=t3;
       System.out.print("t1 =");
       for (int i=0;i<t1.length;i++) System.out.print (t1[i]);
       System.out.println();
       System.out.print("t2 =");
       for (int i=0;i<t2.length;i++) System.out.print (t2[i]);
       System.out.println();
       System.out.print("t3 =");
       for (int i=0;i<t3.length;i++) System.out.print (t3[i]);
       System.out.println();
       }
}

t1=hello
t2=bonjour
t3=bonjour
```

Les paramètres du programme

```
class Prog23 {
  public static void main (String args[])
    {int i;
    System.out.println("les arguments :");
    for (i=0; i<args.length ; ++i)
      System.out.println(args[i]);}
}
```

Exécution

java Prog23 "Bonjour coucou "   a   b
les arguments :
Bonjour coucou
a
b

```
class UtilTab
{static double somme(double[] t)
        {double s=0;     int i;
        for (i=0;i <t.length; i++) s+=t[i];    return s;}

static void affiche (double[] t)
{for (int i=0; i <t.length; i++) System.out.print(t[i]+" ");
System.out.println();}

public static double[] somme (double[] t1, double[] t2)
{       int n=t1.length;
        if (n!=t2.length) return null;
        double [] res= new double[n];
        for (int i=0; i <n; i++) res[i]=t1[i]+t2[i];   return res;}
}
public class TstUtil
{ public static void main(String[] args)
        { double tab1[]={1.5, 2.25, 3.5, 4.0};
        System.out.println("tab1 "); UtilTab.affiche(tab1);
        System.out.println("somme" + UtilTab.somme(tab1));
        double tab2[]={2.5, 1.5, 3.5, 4.5};
        System.out.println("tab2 "); UtilTab.affiche(tab2);
        double [] tab3= UtilTab.somme(tab1,tab2);
        System.out.println("tab1+tab2 ");UtilTab.affiche(tab3);}
}
```

Tableaux à deux dimensions

int [][] tab; la référence tab qui pointe sur null
tab=new int [2][3]; un tableau de taille 2 de
tableaux d'entiers de taille 3 initialisés par défaut à 0

tableau[1][1]=5;

tableau[0]=new int[2];

```
import java.io.*;
import java.util.Scanner;

class Matrice{
int[][] tab;
int nb_ligne, nb_col;

    public Matrice()    {
   int i,j;Scanner keyboard =new Scanner(System.in);
   nb_ligne=keyboard.nextInt();nb_col=keyboard.nextInt();
   tab=new int[nb_ligne][nb_col];
   for (i=0;i<nb_ligne;i++)
   {for(j=0;j<nb_col;j++)
       tab[i][j]=keyboard.nextInt();  }
   }

 public void affiche()
 {int i,j;for (i=0;i<nb_ligne;i++)
 {for (j=0;j<nb_col;j++) System.out.print(tab[i][j]+" ");
 System.out.println(" ");
 } }
```

```
class TableauB
{public static void main(String[] argv)
  {boolean[] tableau={true,false,true};
System.out.println("Deuxieme element de tableau : "+ tableau[1]); }
}

/*A l'execution, on obtient :
Deuxieme element de tableau : false
*/
```

Tableaux d'objets

```
class Point
{private int x,y;

public Point(int a, int b)
{x=a; y=b;}

public int getx()  {return x;}

public int gety()  {return y;}

public void affiche()
{System.out.println("x = "+x+" y = "+y);}
}

class Gestion_tab
{ private Point[] tab;
  private int nb_pts_positifs;

public Gestion_tab( int n)  {tab=new Point[n];}

public void ajout_pts()
{for (int i=0; i <tab.length;i++) tab[i]=new Point();}
```

```
public void calcul_pts_positifs()
{int nb=0, i;
for (i=0;i < tab.length;i++)
        if ((tab[i].getx() >0) &&(tab[i].gety()>0)) nb++;
        nb_pts_positifs=nb;
}

public void affiche()
{for (int i=0; i <tab.length;i++) tab[i].affiche();}
}

public class TestGestion
{ public static void main (String[] args)
        {Gestion_tab t=new Gestion_tab(3);
        t.ajout_pts(); t.affiche();
        t.calcul_pts_positifs();}
}
```

FOR EACH

double[] a=new double[3];

```
for (int i = 0; i < a.length; i++)
        a[i] = 0.0;
```

–équivalent à

```
for (double element : a)
        element = 0.0;
```

```
for (TypedeBaseTableau NomVariable : NomTableau)
{instructions}
```

Méthodes avec un nombre variable d'arguments

```
int... a
Type... ArrayName
```

les arguments sont placés dans un tableau

```
public class UtilityClass
{
    /**
     Returns the largest of any number of int values.
    */
   public static int max(int... arg)
    {
        if (arg.length == 0)
        {
           System.out.println("Fatal Error: maximum of zero values.");
           System.exit(0);
        }

        int largest = arg[0];
        for (int i = 1; i < arg.length; i++)
            if (arg[i] > largest)
                largest = arg[i];
        return largest;
    }
}
```

Classe String

•Définie dans le paquetage java.lang

```
String blessing = "Live long and prosper.";
```

•Méthodes

- public int length();

•public int compareTo(String s);

>0 si la chaîne est après s (ordre lexicographique)

<0 si elle est avant s

=0 si égale

int l= "bonjour ".length();

String s= "abc" ; String s1= "dbc" ;

if (s.compareTo(s1) <0)

- *Concatenation*: +

```
String greeting ="Hello ";
String javaClass ="class";
```

`greeting + javaClass` est `"Hello class"`

concatenation de String avec un autre type est un String

`"The answer is " + 42` est `"The answer is 42"`

Classe StringBuffer

Les chaînes modifiables

- charAt(int i) retourne le ième char
  (classe String a aussi cette méthode)
- setCharAt(int i, char c) remplace le ième char par c

Display 1.4 Some Methods in the Class String

`int length()`

Returns the length of the calling object (which is a string) as a value of type `int`.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.length()` returns 6.

`boolean equals(Other_String)`

Returns `true` if the calling object string and the *Other_String* are equal. Otherwise, returns `false`.

EXAMPLE

After program executes `String greeting = "Hello";`

```
greeting.equals("Hello") returns true
greeting.equals("Good-Bye") returns false
greeting.equals("hello") returns false
```

Note that case matters. `"Hello"` and `"hello"` are not equal because one starts with an uppercase letter and the other starts with a lowercase letter.

(continued)

Display 1.4 Some Methods in the Class `String`

`boolean equalsIgnoreCase(`*Other_String*`)`

Returns `true` if the calling object string and the *Other_String* are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns `false`.

EXAMPLE

After program executes `String name = "mary!";`
`greeting.equalsIgnoreCase("Mary!")` returns `true`

`String toLowerCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toLowerCase()` returns `"hi mary!"`.

(continued)

Display 1.4 Some Methods in the Class `String`

`String toUpperCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to uppercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toUpperCase()` returns `"HI MARY!"`.

`String trim()`

Returns a string with the same characters as the calling object string, but with leading and trailing white space removed. Whitespace characters are the characters that print as white space on paper, such as the blank (space) character, the tab character, and the new-line character `'\n'`.

EXAMPLE

After program executes `String pause = "  Hmm  ";`
`pause.trim()` returns `"Hmm"`.

(continued)

Display 1.4 Some Methods in the Class `String`

`char charAt(`*Position*`)`

Returns the character in the calling object string at the *Position*. Positions are counted 0, 1, 2, etc.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.charAt(0)` returns `'H'`, and
`greeting.charAt(1)` returns `'e'`.

`String substring(`*Start*`)`

Returns the substring of the calling object string starting from *Start* through to the end of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2)` returns `"cdefG"`.

(continued)

Display 1.4 Some Methods in the Class `String`

`int indexOf(A_String, Start)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string that occurs at or after position *Start*. Positions are counted 0, 1, 2, etc. Returns −1 if *A_String* is not found.

EXAMPLE

After program executes `String name = "Mary, Mary quite contrary";`
`name.indexOf("Mary", 1)` returns `6`.
The same value is returned if `1` is replaced by any number up to and including 6.
`name.indexOf("Mary", 0)` returns `0`.
`name.indexOf("Mary", 8)` returns `−1`.

`int lastIndexOf(A_String)`

Returns the index (position) of the last occurrence of the string *A_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns −1, if *A_String* is not found.

EXAMPLE

After program executes `String name = "Mary, Mary, Mary quite so";`
`greeting.indexOf("Mary")` returns `0`, and
`name.lastIndexOf("Mary")` returns `12`.

(continued)

Display 1.4 Some Methods in the Class `String`

`int compareTo(A_String)`

Compares the calling object string and the string argument to see which comes first in the lexicographic ordering. Lexicographic order is the same as alphabetical order but with the characters ordered as in Appendix 3. Note that in Appendix 3 all the uppercase letters are in regular alphabetical order and all the lowercase letters are in alphabetical order, but all the uppercase letters precede all the lowercase letters. So, lexicographic ordering is the same as alphabetical ordering provided both strings are either all uppercase letters or both strings are all lowercase letters. If the calling string is first, it returns a negative value. If the two strings are equal, it returns zero. If the argument is first, it returns a positive number.

EXAMPLE

After program executes `String entry = "adventure";`
`entry.compareTo("zoo")` returns a negative number,
`entry.compareTo("adventure")` returns `0`, and
`entry.compareTo("above")` returns a positive number.

(continued)

•Méthode **toString**

public String toString()

retourne la chaîne de caractères pour les données d'un objet

```
Point a=new Point(1,3);
System.out.println(a);affichera
```

Point@2ac982

si la méthode toString est définie comme suit dans la classe Point:

```
public String toString()
{return("x= "+x+ "  y= "+y);}
```

```
System.out.println(a.toString);   affichera
```

x= 1 y= 3

```
 System.out.println(a);   affichera
```

x= 1 y= 3

ENUMERATIONS

Définis avant les méthodes (avec les constantes)

```
enum TypeName {VALUE_1, VALUE_2, …, VALUE_N};
```

majuscules (constantes)

```
enum WorkDay {MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY}
```

Déclaration des variables

```
WorkDay meetingDay, availableDay;
```

Affectation des valeurs

```
meetingDay = WorkDay.THURSDAY;
availableDay = null;
```

```
WorkDay meetingDay = WorkDay.THURSDAY;
```

Affichage

```
System.out.println(meetingDay);
```

résultat:

```
THURSDAY
```

comme

```
System.out.println(WorkDay.THURSDAY);
```

égalité avec (==) ou (equals)

```
if (meetingDay == availableDay)
   System.out.println(”reunion aura lieu.");
if (meetingDay == WorkDay.THURSDAY)
   System.out.println("weekend prolonge!);
```

Display 6.13 An Enumerated Type

```
1  public class EnumDemo
2  {
3      enum WorkDay {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};

4      public static void main(String[] args)
5      {
6          WorkDay startDay = WorkDay.MONDAY;
7          WorkDay endDay = WorkDay.FRIDAY;

8          System.out.println("Work starts on " + startDay);
9          System.out.println("Work ends on " + endDay);
10     }
11 }
```

SAMPLE DIALOGUE

```
Work starts on MONDAY
Work ends on FRIDAY
```

Display 6.14 Some Methods Included with Every Enumerated Type

`public boolean equals(`*Any_Value_Of_An_Enumerated_Type*`)`

Returns `true` if its argument is the same value as the calling value. While it is perfectly legal to use `equals`, it is easier and more common to use ==.

EXAMPLE

For enumerated types, (*Value1*`.equals(`*Value2*`)`) is equivalent to (*Value1* == *Value2*).

`public String toString()`

Returns the calling value as a string. This is often invoked automatically. For example, this method is invoked automatically when you output a value of the enumerated type using `System.out.println` or when you concatenate a value of the enumerated type to a string. See Display 6.15 for an example of this automatic invocation.

EXAMPLE

`WorkDay.MONDAY.toString()` returns "MONDAY".
The enumerated type `WorkDay` is defined in Display 6.13.

(continued)

Display 6.14 Some Methods Included with Every Enumerated Type

```
public int ordinal()
```

Returns the position of the calling value in the list of enumerated type values. The first position is 0.

EXAMPLE

`WorkDay.MONDAY.ordinal()` returns 0, `WorkDay.TUESDAY.ordinal()` returns 1, and so forth. The enumerated type `WorkDay` is defined in Display 6.13.

```
public int compareTo(Any_Value_Of_The_Enumerated_Type)
```

Returns a negative value if the calling object precedes the argument in the list of values, returns 0 if the calling object equals the argument, and returns a positive value if the argument precedes the calling object.

EXAMPLE

`WorkDay.TUESDAY.compareTo(WorkDay.THURSDAY)` returns a negative value. The type `WorkDay` is defined in Display 6.13.

```
public EnumeratedType[] values()
```

(continued)

Display 6.14 Some Methods Included with Every Enumerated Type

Returns an array whose elements are the values of the enumerated type in the order in which they are listed in the definition of the enumerated type.

EXAMPLE

See Display 6.15.

```
public static EnumeratedType valueOf(String name)
```

Returns the enumerated type value with the specified name. The string `name` must be an exact match.

EXAMPLE

`WorkDay.valueOf("THURSDAY")` returns `WorkDay.THURSDAY`. The type `WorkDay` is defined in Display 6.13.

```
import java.util.Scanner;
public class EnumSwitchDemo
{ enum Flavor {VANILLA, CHOCOLATE, STRAWBERRY};

    public static void main(String[] args)
    { Flavor favorite = null;
        Scanner keyboard = new Scanner(System.in);
        System.out.println("What is your favorite flavor?");
        String answer = keyboard.next();

         answer = answer.toUpperCase();
        favorite = Flavor.valueOf(answer);
        switch (favorite)
        {case VANILLA:
                System.out.println("Classic");
                break;
            case CHOCOLATE:
                System.out.println("Rich");
                break;
            default:
                System.out.println("I suppose you said STRAWBERRY.");
                break;
        }
    }
}
```