Programmation Java

procédures, fonctions et String

P. Bakowski

bako@ieee.org

Procédures

Une procédure est un bout de code qui a un nom et qui peut etre appelé depuis le reste du programme.
Une procedure est forcement dans une classe.

```
public class Lignes1{
public static void main(String args[]){
dessinerLigne();
Terminal.ecrireStringln("Un texte bien encadre");
dessinerLigne();
Terminal.ecrireStringln("Apres la ligne");
Terminal.sautDeLigne();
dessinerLigne();
}
public static void dessinerLigne(){
for(int i=0;i<60;i++){Terminal.ecrireChar('*');}
Terminal.sautDeLigne();}
}
```

Variables locales

```
public class VarLocales {
  public static void dessinerLigne(){
  int a=0;              // variable locale
  while (a<60){
  Terminal.ecrireChar('*');
  a=a+1;
  }
  Terminal.sautDeLigne ( ) ;
  }
  public static void main(String args[]){
  int a=20;
  dessinerLigne();
  Terminal.ecrireIntln(a);
  }
}
```

```
Java/Test>java VarLocales
************************************************************
20
```

Arguments d'une procédure

```
public class ProcedureAvecArguments {
  public static void dessinerLigne(int longueurLigne){
  for (int i=0;i<longueurLigne;i++){
  Terminal.ecrireChar('*');}
  Terminal.sautDeLigne();
  }
  public static void main(String args[]){
  int k=50;
  dessinerLigne(k);
  dessinerLigne(k/2);
  dessinerLigne(k/4);
  }
}
```

Pour appeler une procédure qui prend des arguments,
on écrit le nom de la procédure, suivi des valeurs des arguments,
entre parentèses.

Localité des paramétres

Le passage de paramètre en Java se fait par valeur. Le paramètre est une variable creée lors de l'appel de la procédure, qui reçoit une copie de la valeur qu'on lui passe.
Une modification d'un paramètre ne modifie pas a priori la variable correspondante du programme principal.

```
public class Localite {
public static void augmenterMalEcrit(int val){
val=val+1;
}
public static void main(String args[]){
int k=10;
augmenterMalEcrit(k);
Terminal.ecrireIntln(k);
}
}
```

La valeur affichée est ???

Surcharge

Il serait intéressant de disposer de plusieurs variantes de `dessinerLigne`.
Dans un premier temps, on pourrait imaginer de leur donner tout simplement des noms différents :

```
dessinerLigne();
dessinerLigneDeLongueurDonnee(int k);
dessinerLigneDeLongueurDonneeAvecSymbole(int k,char s);
```

Mais il est plus simple pour le programmeur qui utilise les procédures de ne pas s'encombrer la mémoire avec plusieurs noms.
Pour résoudre ce problème, java utilise la surcharge. La surcharge donne la possibilité de déclarer plusieurs méthodes avec le même nom.
Elles sont alors distinguées par leur liste d'arguments.

```
dessinerLigne();
dessinerLigne(int k) ;
dessinerLigne(int k, char s) ;
```

Surcharge

```
public class LigneMania{
public static void dessinerLigne(int longueur,char symbole){
for(int i=0;i<longueur;i++) {
Terminal.ecrireChar(symbole);}
Terminal.sautDeLigne();}

public static void dessinerLigne(int longueur){
  dessinerLigne(longueur,'*');}

public static void dessinerLigne(){
  dessinerLigne(60,'*');}

public static void main(String args[]){
dessinerLigne();
dessinerLigne(20);
Terminal.sautDeLigne();
dessinerLigne(100,'-');
}
}
```

Passage de tableaux en argument

Pour un tableau passé en paramètre, java passe la référence du tableau (son adresse, en fait). En conséquence, le tableau d'origine et celui vu par la procédure occupent le même espace mémoire. Une première conséquence de cela est que le passage d'un tableau en paramètre n'entraîne pas la copie de celui-ci (Ici : `t` est la même adresse que `tab`).

```
public class TestTab1{
  public static void main(String args[]){
  int tab[]={8,2,1,23};
  afficherTableau(tab);
  }
  public static void afficherTableau(int t[]){
  for(int i=0;i<t.length;i++) {
    Terminal.ecrireInt(t[i]);
    if(i!=t.length-1)Terminal.ecrireChar(' ');
    }
  Terminal.sautDeLigne();
  }
}
```

Procédure dans une autre classe

On peut créer des bibliothèques de procédures.
Vous en utilisez déjà une : la classe `Terminal`.

On pourrait vouloir utiliser notre classe `LigneMania` de la même manière.

Pour utiliser les méthodes d'une classe `A` depuis une classe `B` :
– il faut que le code des deux classes soit dans le même dossier,
– on peut appeler les méthodes de `A` en préfixant simplement le nom de la mêthode par `A.` (Ici : `LigneMania. ).`

```
public class TestLignes {
  public static void main(String args[]){
  LigneMania.dessinerLigne(100,'-');
  }
}
```

Fonctions

Une fonction ressemble a une procédure, mais sa pour tâche est de renvoyer une valeur, qui sera utilisée par la suite. L'exemple le plus immédiat, ce sont les fonctions mathématiques.

La fonction **cos**, de la classe **Math**, calcule ainsi le cosinus de son argument. On l'utilise de la manière suivante :

```
double x= 3;
double y= Math.cos(x/2);
```

D'un point de vue pratique, dans une fonction :
– on met le type de la valeur renvoyée a la place du **void** des procédures :
– on renvoie la valeur du résultat a l'aide du mot-clef **return** .

Fonctions

```
public class F1 {
  public static double valeurAbsolue(double x){
  double resultat;
  if(x<0)
    resultat=-x;
  else
    resultat= x;
  return resultat ;
  }
  public static void main(String args[]){
  double v=Terminal.lireDouble();
  Terminal.ecrireDoubleln(valeurAbsolue(v));
  }
}
```

La fonction `valeurAbsolue` déclare dans son en-tête qu'elle retourne une valeur de type **`double`**. La valeur est effectivement renvoyée a l'aide de la commande **`return`**.

Fonctions qui retournent un tableau

Une fonction peut très bien renvoyer un tableau. Il suffit pour cela qu'elle le déclare comme type de retour.
La fonction suivante renvoie par exemple un tableau, dont la taille et le contenu des cases sont donnés en paramètre.

```
public static double[]
  creerTableau(int taille, double valeur) {
  double[] t=new double[taille]; // creation du tableau
  for(int i=0;i<taille;i++)
  t[i]= valeur;
  return t ;  // retour de l'adresse du tableau - t
}
```

Chaînes de caractères

Quatre moyens pour créer une chaîne (String) :
– en ecrivant cette chaîne avec ses guillemets dans le programme.
– au moyen de l'instruction new.
– au moyen de l'opérateur + :
  le résultat d'une concaténation est une nouvelle chaîne
– au moyen d'une méthode qui renvoie une nouvelle chaîne.

```
String s = "bonjour";

char[] tab = {'b','o','n','j','o','u','r'};
String s = new String(tab);
String s =
new String(new char[]{'b','o','n','j','o','u','r'});
```

Méthode `length()`

On peut appeler `length` non seulement sur une variable mais sur d'autres sortes d'expressions calculant un objet String :
– une chaîne entre guillement : `"bonjour".length()`
– une chaîne créée par `new` : `(new String()).length()`
– le résultat d'une concaténation : `(s + "qsd").length()`

```
String s = "bonjour";
Terminal.ecrireIntln(s.length());
```

Tableaux et chaînes

Les points communs:

– il y a deux temps différents : la déclaration et la création d'une valeur.
– après la déclaration une variable contient la valeur **`null`**.
– la création explicite d'une nouvelle chaîne au moyen d'une instruction **`new`**.
– il y a possibilité de création implicite au moyen d'une syntaxe spéciale.
  Les tableaux - les accolades `{..}`, les chaînes - les guillemets «`..`».
– les chaînes et les tableaux ont une longueur supérieure ou egale a 0.
  Elle est fixe et invariable dans le temps.

Contrairement aux tableaux :

– il n'y a pas une syntaxe spécifique pour accéder directement aux caractères d'une chaîne de caractères.

Les chaînes de caractères sont le premier exemple d'objet dans ce cours.
Les types des objets sont comme les tableaux des types références, c'est a dire que les variables de ces types contiennent l'adresse des objets ou des tableaux en mémoire.

Quelques méthodes sur `String`

```
public class ExChaine2 {
  public static void main(String args[]){
  String s1 = "bonjour";
  String s2 ;
Terminal.ecrireString("Entrez une chaine:");
s2 = Terminal.lireString();
Terminal.ecrireCharln(s1.charAt(0));
Terminal.ecrireStringln(s1.toUpperCase());
Terminal.ecrireIntln(s1.compareTo(s2));
Terminal.ecrireStringln(" "+s1.equals(s2));
}
}
```

A noter : il n'y a pas de méthode ni d'autre moyen de changer un caractère dans une chaîne : une fois la chaîne créee, on ne peut pas la modifier. Pour obtenir un résultat plus ou moins equivalent a un changement de caractère, il faut créer une nouvelle chaîne en concaténant des portions de la chaîne originale avec le caractère différent.

Comparaison de chaînes

```
public class EgCh{
public static void main(String args[]){
String s1,s2,s3;
boolean b1,b2;
s1 = "tati";  s2 = "ta";
s2 = s2 + "ti";  s3 = s1 ;
Terminal.ecrireStringln("s1="+s1+"s2="+s2+"s3="+s3);
b1 = (s1==s2);
b2 = (s1==s3);
Terminal.ecrireStringln("s1==s2?"+b1);
Terminal.ecrireStringln("s1==s3?"+b2);
}
}
```

Comme pour un tableau, les opérateurs == et `!=` ne regardent pas le contenu des chaînes mais juste leur adresse en mémoire.
La question posée est :

les deux chaînes sont-elles à la même adresse ?

Paramètre de la méthode `main()`

```
public class LigneCommande{
  public static void main(String args[]){
    for(int i=0;i<args.length;i++){
      Terminal.ecrireStringln(args[i]);
    }
  }
}
```

```
Java/Test> java LigneCommande un 12 56 deux
un
12
56
deux
```

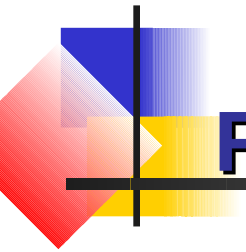
Conversion entre chaînes et autres

```
public class StringInt2{
  public static void main(String args[]){
    int x ;
    String s= «12»;
    x = Integer.parseInt(s);
    Terminal.ecrireIntln(x);
  }
}
```

Pour convertir une valeur de type **`double`**, il faut utiliser la méthode `Double.parseDouble(s)` et pour le type **`boolean`**, la méthode `Boolean.parseBoolean(s)`.

Pour convertir dans l'autre sens, un `int` en chaîne, le plus simple est d'utiliser l'opérateur de concaténation :
`""+12` (pour les doubles `""+12.3`, pour les booléens `""+true`).

On concatène la chaîne vide avec la valeur à convertir.



Résumé

Procédures sans et avec arguments

Surcharge

Fonctions

Chaînes – `String`

Conversion entre chaînes et autres types