Cours Introduction à la Programmation
Java V
(IP1 JAVA)

Arnaud Sangnier
sangnier@irif.fr

Mercredi 15 Novembre 2017
INFO et MATHS-INFO

Dans les épisodes précédents

- Présentation de certains aspects de Java
- Ce que nous avons vu :
  - Les données et leur type : int , String et boolean
  - Les variables : affectation, lecture, modification
  - Toutes les instructions
  - Définition de fonctions
  - **Les tableaux à une dimension et leur utilisation basique**

Retour sur le passage des arguments

- Prenons ce programme

```
public static int f(int a) {
  a = a+1;
  return a;}

… main(...){
  int x = 3;
  int y = f(x);
  System.out.println(x);
  System.out.println(x);}
```

- Quand on appelle int y = f(x), la valeur de x est donnée à f, mais f ne modifie pas x
- Ce programe affiche 3 et 4
- On parle de passage par valeurs

Que se passe-t-il avec les tableaux ?

- Prenons ce programme

```
public static f(int[]t){
  t[0]=2;
  return t;}

… main(...){
  int[] tab =new tab[2];
  tab[0]=0;
  tab[1]=0;
  int[] tab2=f(tab);
  System.out.println(tab[0]);
  System.out.println(tab2[0]);}
```

- Qu'affiche ce programme ? 2 et 2
- Pourquoi ? Que donne-t-on exactement comme arguments à f au moment de l'appel ?

Il faut comprendre à quoi correspondent les variables de tableaux

- En fait quand on fait : int[] tab=new int[2];
- Ce qui est stocké dans la variable tab de type int[], ce n'est pas le tableau {0.0} mais c'est son 'adresse' en mémoire
- Ainsi si on veut copier un tableau c'est à dire le créer deux fois on ne peut pas faire :

```
int[] t=new int[3];
int[] t2=t;
```

- Ici les deux variables t et t2 indiquent le même tableau

Retour sur l'utilisation des tableaux

- Quand on fait int[] t=new int[10]; → on réserve 10 cases dans la mémoire à une certaine adresse que l'on stocke dans t
- t contient donc une adresse
- Si on fait t[i] on dit au programme d'aller voir dans la i-ème case située à l'adresse stockée dans t
- Quand on fait t2=t , t2 reçoit la même adresse que t, mais il n'y a toujours eu qu'un seul tableau de créé

Exemple

- Prenons ce programme

```
… main(...){
  int a = 5;
  int b = a;
  b = b +1;
  System.out.println(a);
  System.out.println(b);}
```

- a reçoit la valeur 5
- b reçoit la valeur de a qui vaut 5
- b augmente de 1 sa valeur
- à la fin a vaut 5 et b vaut 6

Exemple

- Prenons ce programme

```
… main(...){
  int[] tab=new int[6];
  int[] tab2 = tab;
  tab[1]=0;
  tab2[1]=3;
  System.out.println(tab[1]);}
```

- tab reçoit l'adresse $1 du tableau {?,?,?,?,?,?} créé
- tab2 reçoit la valeur $1
- il modifie la deuxième valeur du tableau situé à tab qui vaut $1
- il modifie la deuxième valeur du tableau situé à tab2 qui vaut $1
- le programme affiche la valeur de tab[1], comme tab vaut $1 alors le programme affiche 3
- Pourquoi avoir écrit $1 ? En fait les adresses sont souvent des nombres en binaires choisi par le programme au cours de l'exécution

Exemple

- Prenons ce programme

```
… main(...){
  int[] tab=new int[6];
  int[] tab2 = new int[6];
  tab[1]=0;
  tab2[1]=3;
  System.out.println(tab[1]);}
```

- tab reçoit l'adresse $1 du tableau {?,?,?,?,?,?} créé
- tab2 reçoit la valeur $2 du tableau {?,?,?,?,?,?} créé
- il modifie la deuxième valeur du tableau situé à $2
- le programme affiche la valeur de tab[1], comme tab vaut $2 alors le programme affiche 0
- Ici deux tableaux sont créés et ils sont par conséquent à des adresses différentes

Retour sur le passage en arguments

- Prenons ce programme

```
public static f(int[]t){
  t[0]=2;
  return t;}

… main(...){
  int[] tab =new tab[2];
  tab[0]=0;
  tab[1]=0;
  int[] tab2=f(tab);
  System.out.println(tab[0]);
  System.out.println(tab2[0]);}
```

- Quand on fait l'appel f(tab),on met dans la variable t de la fonction f l'adresse stockée dans tab
- Au final dans ce programme il n'y a qu'un seul tableau
- **Quand on modifie un tableau dans une fonction, ce tableau est aussi modifiée quand on sort de la fonction**

Exemple

- Prenons le programme suivant :

```
public static int g(int[] t) :
  int c = 0 ;
  for(int i=0;i<t.length;i=i+1){
    c = c + t[i] ;
    li[i]=0 ;}
  return c ;}

… main(...){
 int[] tab = {1,2,3,4,5,6} ;
 int a = g(tab) ;
 System.out.println(tab[0]) ;
 System.out.println(a) ;}
```

- Il affiche 0 et 21
- La modification du tableau dans la fonction g se voit à l'extérieur de la fonction

Bien comprendre ce que doivent faire les fonctions

- Donc les fonctions peuvent modifier les tableaux passés en arguments
- Il faut faire la différence :

  1) Écrire une fonction qui prend en argument un tableau d'entiers et décale ses éléments de 1 vers la droite

  2) Écrire une fonction qui prend en argument un tableau d'entiers et renvoie un nouveau tableau d'entiers qui correspond au premier tableau avec les éléments décalés de un vers la droite

- Dans le cas 1) le tableau donné en arguments est modifié
- Dans le cas 2) le tableau donné ne doit pas être changé
- En fait dans le cas 1), on peut même faire une fonction qui ne renvoie rien → une procédure qui ne fait que modifier le tableau

Cas 2)

- Écrire une fonction qui prend en argument un tableau d'entiers et renvoie un nouveau tableau d'entiers qui correspond au premier tableau avec les éléments décalés de un vers la droite

```
public static int[]shift(int[] ta){
 int[] newt=new int[ta.length] ;
  newt[0]=ta[ta.length-1];
  for(int i=ta.length-1;i>0;i=i-1){
    newt[i]=ta[i-1];}
  return newt;}

...main(){
 int[] t = {1,2,3,4} ;
 int[] t2 =shift(t) ;
}
```

- À la fin de ce programme le tableau indiqué par t sera {1,2,3,4} et celui indiqué par t2 sera {4,1,2,3}

Cas 1)

- Écrire une fonction qui prend en argument un tableau d'entiers et décale ses éléments de 1 vers la droite

```
public static int[]shift(int[]ta){
  int tmp=ta[ta.length-1] ;
  for(int i=ta.length-1;i>0;i=i-1){
    ta[i]=ta[i-1];}
  ta[0] = tmp;
  return ta;}

… main(...){
  int[] t = {1,2,3,4} ;
  int[] t2=shift(t)
}
```

- À la fin de ce programme t et t2 indiquent le **même** tableau {4,1,2,3}
- En fait le return ne sert à rien dans ce cas

Cas 1)

- Écrire une fonction qui prend en argument un tableau d'entiers et décale ses éléments de 1 vers la droite

```
public static void shift(int[]ta){
  int tmp=ta[ta.length-1] ;
  for(int i=ta.length-1;i>0;i=i-1){
    ta[i]=ta[i-1];}
  ta[0] = tmp;}

… main(...){
  int[] t = {1,2,3,4} ;
  shift(t)
}
```

- À la fin de ce programme t le tableau {4,1,2,3}

Cas 1) Erreur classique

- Écrire une fonction qui prend en argument un tableau d'entiers et décale ses éléments de 1 vers la droite

```
public static void shift(int[]ta){
  int tmp=ta[ta.length-1] ;
  for(int i=1;i<ta.length;i=i+1){
    ta[i]=ta[i-1];}
  ta[0] = tmp;}

… main(...){
  int[] t = {1,2,3,4} ;
  shift(t)
}
```

FAUX

- À la fin de ce programme t le tableau {4,1,1,1}

Un autre problème classique

- Il faut faire la différence :

  1) Écrire une fonction qui prend en argument un tableau d'entiers et renverse l'ordre de ses éléments

  2) Écrire une fonction qui prend en argument un tableau d'entiers et renvoie un nouveau tableau qui correspond au premier tableau renversé

- Dans le cas 1) le tableau donné en arguments est modifié
- Dans le cas 2) le tableau donné ne doit pas être changé
- En fait dans le cas 1), on peut même faire une fonction qui ne renvoie rien → une procédure qui ne fait que modifier le tableau

Cas 2)

- Écrire une fonction qui prend en argument un tableau d'entiers et renvoie un nouveau tableau qui correspond au premier tableau renversé

```
public static in[] reverse(int[] ta){
  int[] newt=new int[ta.length] ;
  for(int i=0;i<ta.length;i=i+1){
    newt[i]=ta[ta.length-1-i];}
  return newt;}

…. main(...){
  int[] t= {1,2,3,4} ;
 int[] t2=reverse(t)
}
```

- À la fin de ce programme le tableau indiqué par t sera {1,2,3,4} et celui indiqué par t2 sera {4,3,2,1}

Cas 1)

- Écrire une fonction qui prend en argument un tableau d'entiers et renverse l'ordre de ses éléments

```
public static void reverse(int[] ta){
  for(int i=0;i<ta.length;i=i+1){
    ta[i]=ta[ta.length-1-i];}
}

…. main(...){
  int[] t= {1,2,3,4} ;
 reverse(t)
}
```

- À la fin de ce programme, t indique le tableau {4,3,2,1}

Cas 1)

- Écrire une fonction qui prend en argument un tableau d'entiers et renverse l'ordre de ses éléments

```
public static void revers(int[] ta){
  for(int i=0;i<ta.length;i=
    ta[i]=ta[ta.length-1-i];}
}

...ain(...){
 nt[]  {1,2,3,4
 everse
```

FAUX

- À la fin de le programme, t indique le tableau {4,3,2,1}

Cas 1)

- Écrire une fonction qui prend en argument un tableau d'entiers et renverse l'ordre de ses éléments

```
public static void reverse(int[] ta){
  for(int i=0;i<ta.length;i=i+1){
    ta[i]=ta[ta.length-1-i];}
}

…. main(...){
  int[] t= {1,2,3,4} ;
 reverse(t)
}
```

- **À la fin de ce programme, t indique le tableau {4,3,3,4}**

Cas 1)

- Écrire une fonction qui prend en argument un tableau d'entiers et renverse l'ordre de ses éléments

```
public static void reverse(int[] ta){
  for(int i=0;i<ta.length/2;i=i+1){
    int tmp=ta[i ;
    ta[i]=ta[ta.length-1-i];}
    ta[ta.length-1-i]=tmp ;
}

…. main(...){
  int[] t= {1,2,3,4} ;
 reverse(t)
}
```

- **À la fin de ce programme, t indique le tableau {4,3,2,1}**

Une autre vision classique sur les tableaux

- On a vu que les tableaux et les boucles étaient fortement liées
- Par exemple on peut remplir un tableau ta de taille k en faisant for(int i=0;i<k;i=i+1) et à chaque tour de boucle on accède à la position ta[i]
- Mais ce n'est pas toujours aussi simple.
- Par exemple, écrire une fonction multiple qui prend en arguments un tableau d'entiers ta et un entier k et qui renvoie un tableau contenant les éléments de ta multiples de k
- Ici on doit parcourt ta pour chercher les multiples mais à chaque tour de boucle on ne doit pas ajouter un élément au nouveau tableau

Solution

- Il faut utiliser pour remplir le nouveau tableau une variable p indiquant la position à remplir dans le nouveau tableau
- Au début p vaut 0, et à chaque fois que l'on met un nouvel élément à la position p, on augmente p de 1

```
public static int[] multiple(int[]ta ,int k) {
  int c=0 ;
  for(int i=0;i<ta.length;i=i+1){
    if(ta(i)%k==0){c=c+1;}
  }
  int[] newt=new int[c] ;
  int p =0
  for(int i=0;i<ta.length;i=i+1){
    if(ta(i)%k==0){
      newt[p] = ta[i] ;
      p=p+1 ;
  }
  return newt ;
}
```