Rappels sur les éléments de base de la programmation

Albert DIPANDA

(Bur. R6 aile H)

Structure d'un programme

Une unité de compilation se compose de trois parties:

- L'entête:

  permet de définir (désigner) l'unité de programme

- La partie « déclarations»:

  contient la définition les données

- Le corps du programme:

  contient les instructions à réaliser

Structure d'un programme (Entête)

Une unité de compilation JAVA est définie sous la forme d'une classe

Mot clé: public class nom_classe

Un programme JAVA est composé soit:

- D'une unité de compilation unique
- D'une collection d'unités de compilation contenues dans des fichiers différents d'extension .java
  - Le fichier a le même nom que la classe

Au moins l'une des classes contient une méthode unique dont la signature (ou l'entête) est:

public static void main (String [] args)

Structure d'un programme (Entête)

```
public class essai
{ public static void main(String args[])
  { int[][] t=new int[3][3];
    int[]tt=new int[5];
    //System.out.print("bonjour  ");
    remplitab(t);
    afftab(t);
    int a=1;
    int b=2;
    System.out.print(gagnant(t,a,b));
    remplitab1(tt);
    afftab1(tt);
  }
```

Structure d'un programme (les déclarations)

- Buts:
  - Choix de la représentation des données du programme
  - Réservation de l'emplacement mémoire des données
- Syntaxe: <type> <nom>

Imposé par le langage(java) — Défini par l'utilisateur

Numériques
- Byte (1o)
- Short (2o)
- Int (4o)
- Long (8o)
- Float (4o)
- Double (8o)

Alphanumériques
- Char (2o)
- String

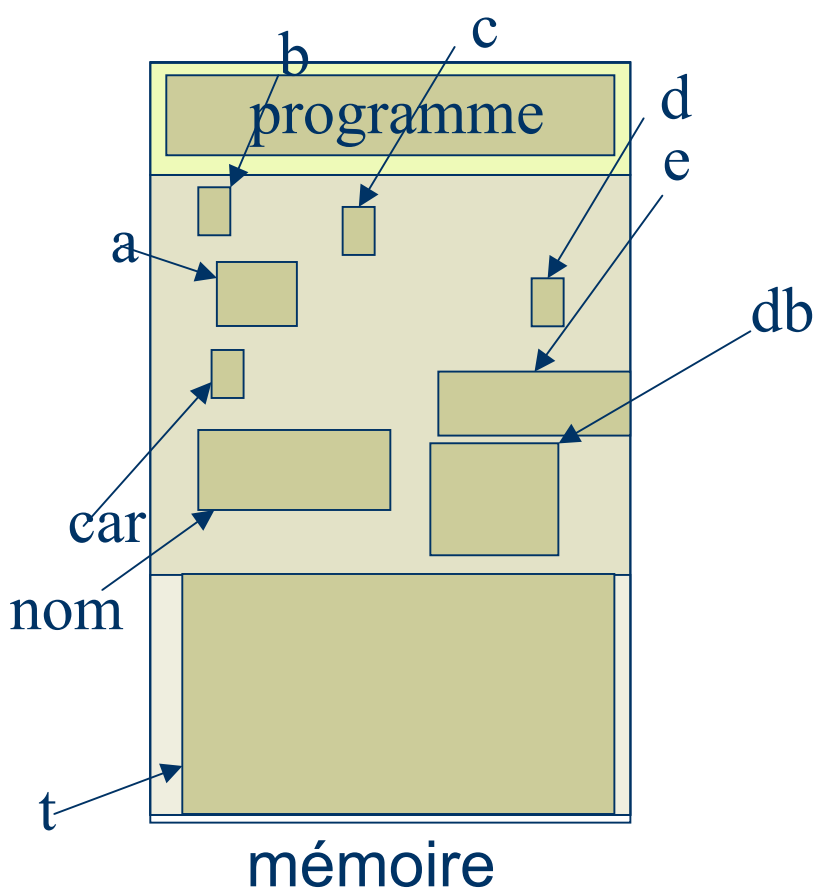
Logiques : boolean (1o)

Tableaux : ensemble de données

Structure d'un programme (les déclarations)

Exemples:

```
public class essai
{ public static void main(String args[])
  {
Int   a=25;
Byte          b, c=a, d=(byte)a;
Long e=a;
Double db=0;
Char car;
String nom='jean';
Int[] t;
…..
db=Lire.d();
……..
t=new int[a]; //Construction du tableau (allocation dynamique)
……..
```



Structure d'un programme (corps du programme-les instructions)

Types d'instructions:

1. Les instructions simples
    1. Les opérations arithmétiques
    2. Les opérations booléennes (logiques)
    3. L'affectation
    4. Les entrées/sorties (saisie et affichage)
2. Les structures de contrôle
    1. Les conditionnelles
    2. Les boucles
    3. Les ruptures de séquence
3. Les appels de fonctions

Structure d'un programme
(corps du programme-les structures de contrôle)

Les conditionnelles

1. if (condition)
   {bloc1}
   else {bloc2} // *pas obligatoire*

2. switch (e)
   {Case e1: {bloc1} break;

   Case en: {blocn} break;
   default : {bloc(n+1)} *//pas obligatoire*
   }

e: type scalaire (int, short, byte, char)

Structure d'un programme
(corps du programme-les structures de contrôle)

Les répétitives

1. **for** (init;condition;incrémentation)

   {….}

   le nombre de répétitions (itérations) est imposé et géré automatiquement par une variable de contrôle.

2. **while** (condition)

   {….}

3. **do**

   {…} **while** (condition)

Le nombre d'itérations dépend de la condition.

Structure d'un programme
(corps du programme-les structures de contrôle)

La rupture de séquence:

- break

  permet la sortie dans un bloc (case, for, while)

- continue

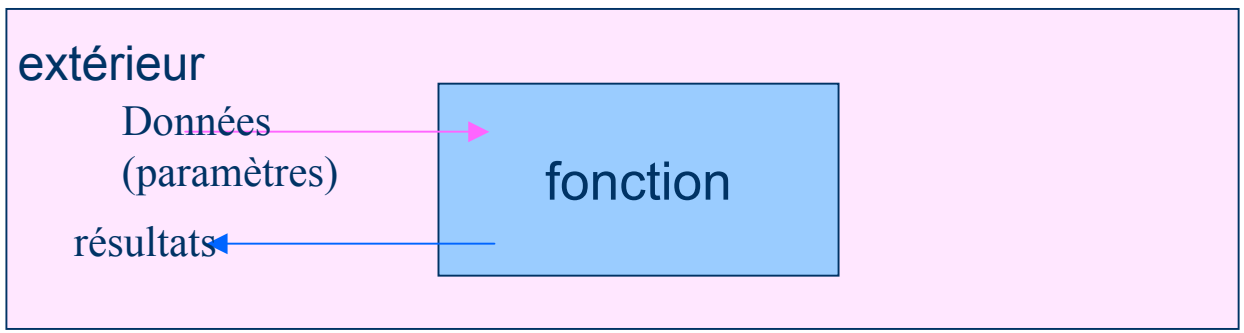
  interrompt le déroulement normal de la séquence

Structure d'un programme (corps du programme-les fonctions)

Une fonction est un programme pouvant être utilisé par d'autres programmes

Réalise un calcul ou une action

Entête: <type_de_retour> <nom_fct> (liste des paramètres)
ou <void>



Question: comment trouver les paramètres?

Réponse: Que fait la fonction? (de quelles données a t-elle besoin?)

Structure d'un programme (corps du programme-les fonctions)

Deux étapes pour l'utilisation de fonctions:

1. Définition de la fonction (indépendant)
   - recensement des données nécessaires à l'utilisation (paramètres formels)
   - définition du corps de la fonction (variables annexes et instructions)
   - Ne pas oublier le « return » en cas de calcul. Doit absolument être accessible

2. Appel de la fonction (dans un programme)
   - se fait en spécifiant le nom de la fonction et des paramètres effectifs qui remplacent les paramètres formels (valeurs ou variables)
   - provoque l'exécution des instructions du corps de la fonction

Structure d'un programme (corps du programme-les fonctions)

Exemples:

Fonction somme : calcule la somme de 2 nombres

- fournit un résultat
- 2 valeurs en donnée (2 paramètres)

Fonction maximum: Recherche et renvoie la plus grande valeur d'un tableau

- fournit un résultat
-un tableau en donnée (paramètre)

Fonction afftab: affiche les valeurs d'un tableau

- ne fournit pas de résultat (void)
- un tableau en donnée (paramètre)

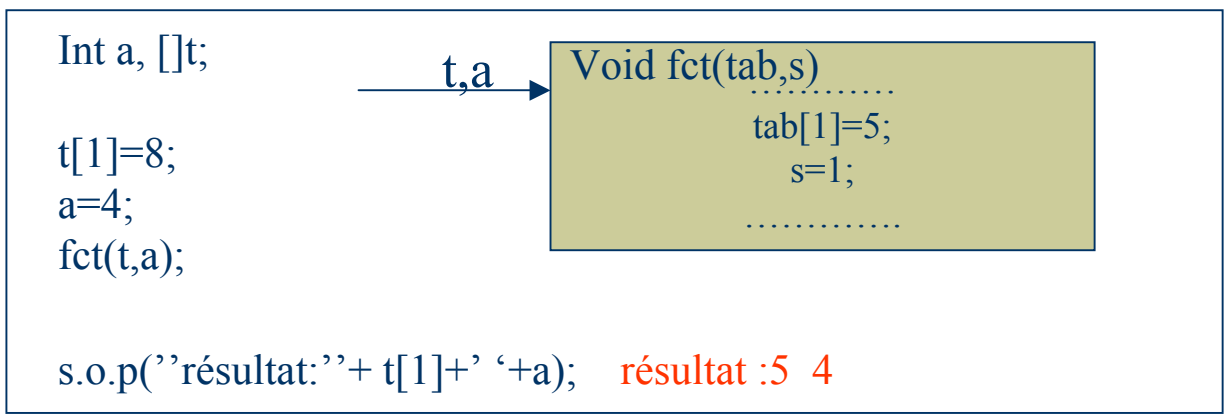
Structure d'un programme (corps du programme-les fonctions)

Question: Que se passe t-il quand un paramètre est modifié dans une fonction?

Réponse:

Si le paramètre est un tableau alors les modifications sont prises en compte à l'extérieur (dans le programme appelant) à la sortie de la fonction

Sinon elles ne sont pas prises en compte



Durée de vie (ou visibilité) des variables

Règle: Une variable est visible (ou *vivante*) dans la « zone » où elle a été déclarée

```
public class Exemple_Visibilite
{static int tout; //variable de classe
  public static void main(String args[])
   { int[] t=new int[5];int i;
      tout=0;
      for(int j=0;j<5;j++)//variable de bloc
            t[j]=Lire.i();
     tout=somtab(t);
      System.out.println(tab[i]);
  }
  public static int somtab(int [] tab)
  {int s; //variable locale
    for(int i=0;i<5;i++) //variable de bloc
      {System.out.println(tab[i]);
       s=s+tab[i];}
      System.out.println(t[1]+' '+tout);
      return s;
   }
}
```

Bloc Exemple_Visibilite

Bloc main

Bloc for

Bloc somtab

Bloc for

Quelques pièges

Gestion des indices (valeurs initiales et finales)

- variable de contrôle d'une boucle for
- parcours de tableaux

```
int i,n;
 int [] t;
n=lire.i();
t=new [n]int;
t[0]=1;
for(i=0;i<n;i++)
t[i++]=t[i]+1;
```

```
for(i=n-1;i>0;i--)
t[i+1]=t[i];
```

```
for(i=n-1;i>0;i--)
t[i++]=t[i];
```

```
for(i=0;i<n;i++);
t[i]=Lire.i();
```

```
s=0;
While (i<n)
  {s=s+t[i];}
s.o.p('la somme est'+s);
```

Quelques pièges

Conditions d'arrêt (boucle while)

```
Int trouve=0;
While (trouve==0)
{If (t[i]==a) trouve=1;
              else i++;
}
s.o.p('la position est'+i);
```

```
While (trouve==0)||(i<n)
{If (t[i]==a) trouve=1;
  else i++;
}
s.o.p('la position est'+i);
```

```
Do
{i++;
If (t[i]==a) trouve=1;
} While (trouve==0)&& (i<n);
s.o.p('la position est'+i);
```

```
While (trouve==0)&& (i<n)
{If (t[i]==a) trouve=1;
  i++;
}
s.o.p('la position est'+i);
```

```
While (i<n)
{int s=0;
  s=s+t[i];
}
s.o.p('la somme est'+s);
```

Donner le résultat de l'exécution

```
public class essai
{ public static void main(String args[])
  { int[] t=new int[5];
    short i;
     for(i=0;i<5;i++);
     System.out.println(i+' ');
     {System.out.println(somme(i,i));}
  }

public static int somme(int a, int b)
 {return (a+b);
}
 }
```

Récursivité

Une fonction est **Récursive** si sa définition contient un (ou plusieurs) appel(s) à lui-même

- **Récursivité directe:** l'appel est fait dans la définition de la fonction
- **Récursivité indirecte :** l'appel est fait dans une autre fonction qui est appelé lors de la définition
- Remarque : dans certains cas la récursivité indirecte est effective au bout de plusieurs appels.

Récursivité

A. Récursivité directe

```
f1 (   )
   {..
     ..
     f1( )
   }
```

Récursivité

B. Récursivité indirecte

```
f1 ( )
   {
      ..
      f2( )
   }

f2 ( )
   {
      ..
      f1( )
   }
```

```
f1 ( )
   {
      f2( )
   }
f2 ( )
   {
      f3( )
   }
f3 ( )
   {
      f1( )
   }
```

Finitude d'une fonction récursive

“ être sûr que le programme se termine au bout d'un nombre fini d'appels ”.

- un algorithme récursif présente deux parties :
  - Un **cas de base** dans lequel il n'est pas fait d'appel récursif,
  - Un **cas général** avec un ou plusieurs appels récursifs mais sur des arguments de "taille" de plus en plus petite jusqu'à atteindre le cas de base.

    notion de taille (valeur d'un argument, longueur d'une liste,...).

- l'existence d'un «**paramètre» de contrôle** entier et positif qui décroît strictement à chaque appel récursif

Finitude d'une fonction récursive

Forme générale d ’une fonction récursive

```
{
    Si <condition>
        Alors <bloc1>{cas de base}
        Sinon <bloc2>{cas général}
    Fsi
}
```

Mise en oeuvre d'une fonction récursive

La méthode d'analyse d'un algorithme récursif comporte trois étapes :

1° Paramétrage du problème : doit permettre de trouver le paramètre de contrôle sur lequel s'applique le programme;

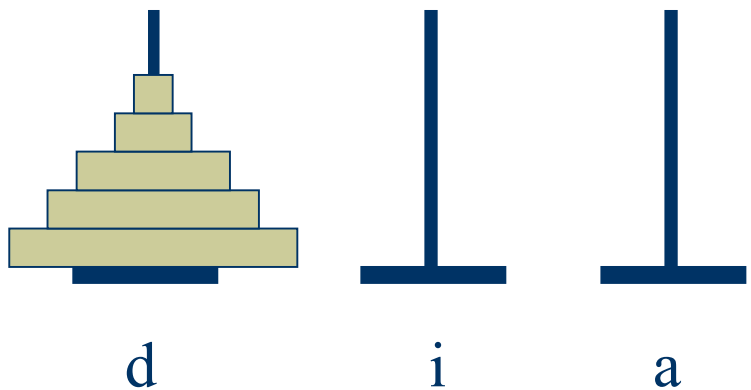
2° Recherche du cas de base — où aucun appel récursif n'est fait. La taille du paramètre est fixée (valeur =0 ou 1, liste vide,...) ;

3° Décomposition du cas général en sous-problèmes similaires mais de taille plus petite.

Exemple d'algorithme récursif

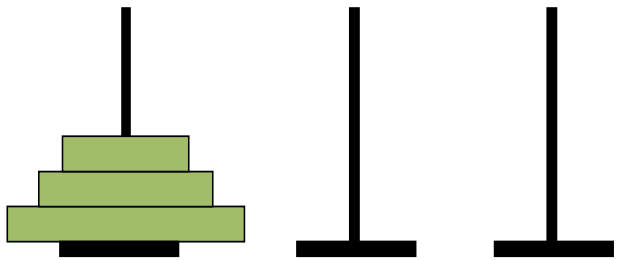
```
Hanoï(n,d,a,i)
{
 if (n==1) //cas de base
    Déplace(d,a);
  else {//sous-cas du cas général
    Hanoï(n-1,d,i,a);
    Déplace(d,a);
    Hanoï(n-1,i,a,d);
  }
}
```

Tours de Hanoi



Tours de Hanoi

(1) Hanoi(3,d,a,i)



(1) Hanoi(3,d,a,i)

(1-1) Hanoi(2,d,i,a)

Deplace(d,a)

(1-2) Hanoi(2,i,a,d)

(1-1) Hanoi(2,d,i,a)

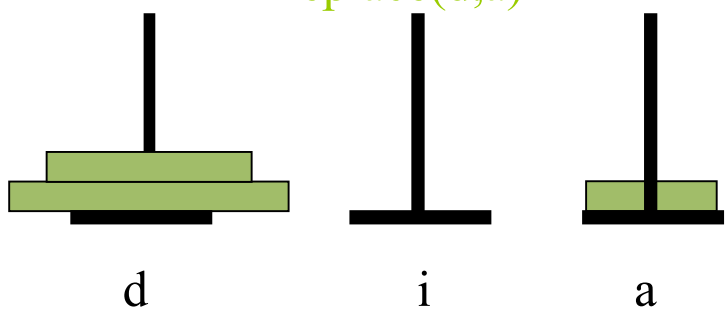
(1-1-1) Hanoi(1,d,a,i)

Deplace(d,i)

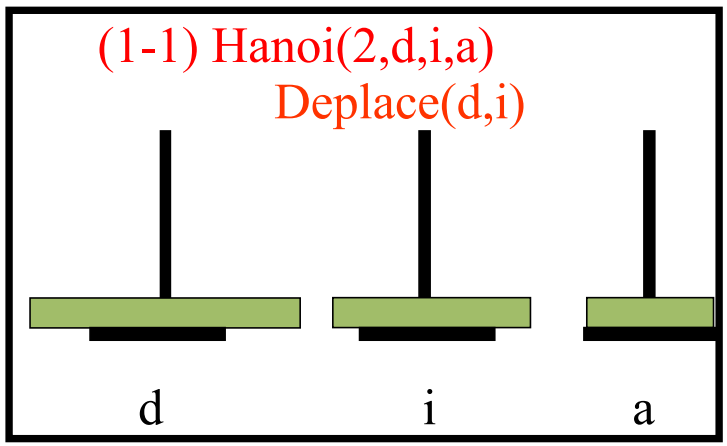
(1-1-2) Hanoi(1,a,i,d)

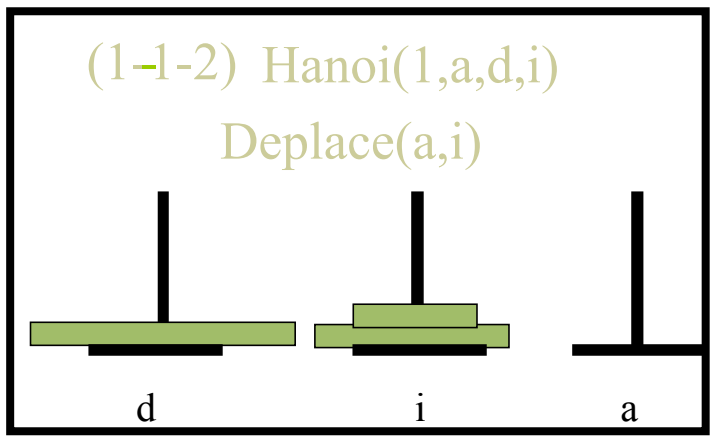
(1-1-1) Hanoi(1,d,a,i)

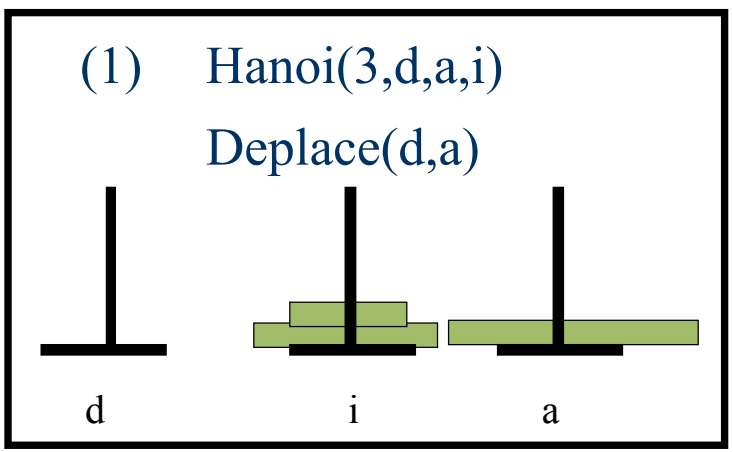
Deplace(d,a)

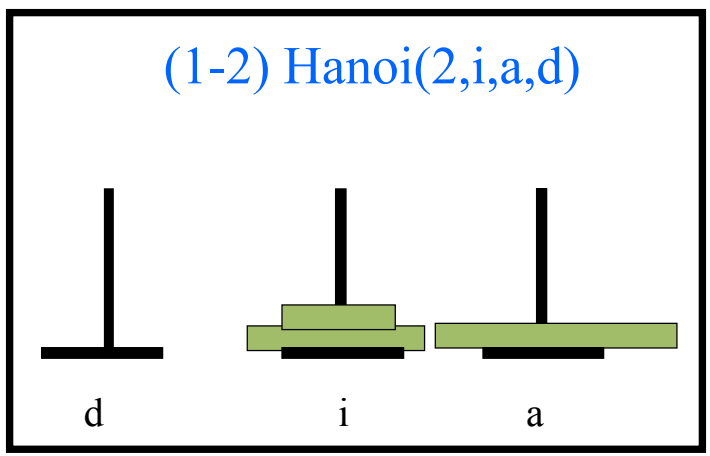


Tours de Hanoi



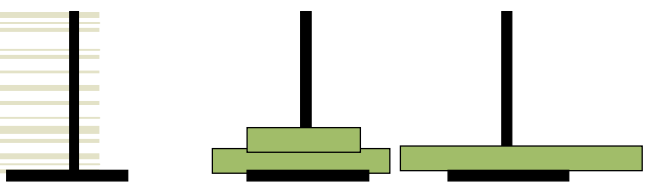






Tours de Hanoi

(1-2) Hanoi(2,i,a,d)



(1-2) Hanoi(2,i,a,d)

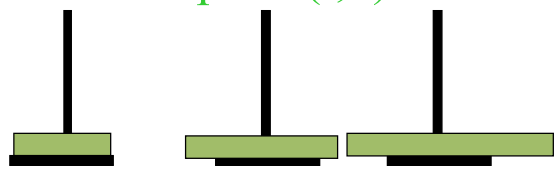
(1-2-1) Hanoi(1,i,d,a)

Deplace(i,a)

(1-2-2) Hanoi(1,d,a,i)

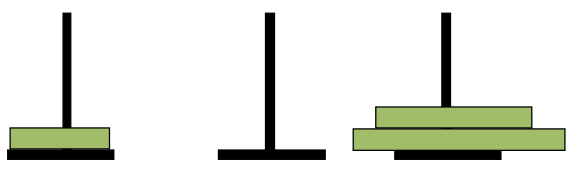
(1-2-1) Hanoi(1,i,d,a)

Deplace(i,d)



(1-2) Hanoi(2,i,a,d)

Deplace(i,a)



(1-2-2) Hanoi(1,d,a,i)

Deplace(d,a)