Fiche de révision JAVA

I-Les types

Entier : int nomdevar ;

Décimaux : double nomdevar ;

Flottant : float nomdevar ;

Caractère : char nomdevar ;

Chaine de caractères : String nomdevar ;

II-Les tableaux

Initialisation d'un tableau : Type [] nomdutab = new type[dimdutab];

Exemple : création d'un tableau d'entier vide de dimension 5

int [] nomdutab=new int[5] ;

Création d'un tableau d'entier de dimension 5 avec 5 valeurs prédéfinies

int [] nomdutab={5,6,8,10,-10000} ;

 **METHODE D'AFFICHAGE D'UN TABLEAU (IMPORTANT)**

```
for(int i=0 ; i<tab.length ;i++){
System.out.println("La case "+i+" de votre tableau est : "+tab[i]);
}
```

 **METHODE GENERALE DE PARCOURT D'UN TABLEAU (IMPORTANT)**

nomdutab[lacasequel'onveuxmodifier] ; (un tableau commence à la case 0 !!!!!)

Boucle for permettant de se déplacer de manière automatique sur toutes les cases du tableau.

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

III-Les méthodes

Une méthode est un morceau de code permettant d'éviter les répétitions. Par exemple, si nous avons un programme avec 5 tableaux et que nous voulons la somme des éléments de chaque tableau, nous pouvons faire ce programme 5 fois :

```
int [] tab={5,3,2,1,-5};
int somme=0;
for (int i=0 ;i<tab.length;i++)
{
        somme+=tab[i];
}
System.out.println(somme);
```

Ou alors nous pouvons créer une méthode que l'on appellera quand nous voudrons faire la somme des éléments d'un tableau comme ceci :

```
public static int somme (int tab[]){
                int somme=0;

                for (int i=0 ;i<tab.length;i++){
                        somme+=tab[i];
                }
                return somme;
        }


public static void main(String[] args) {

int [] tab1={5,3,2,1,-5};
                int [] tab2={6,-3,-2,-1,5};
                int [] tab3={1,2,5,-5,-3};
                int [] tab4={3,3};
                int [] tab5={8,3,2};

                int s1=somme(tab1);
                int s2=somme(tab2);
                int s3=somme(tab3);
                int s4=somme(tab4);
                int s5=somme(tab5);

                System.out.println("s1:"+s1+", s2:"+s2+", s3:"+s3+", s4:"+s4+",
s5:"+s5);


}
```

Cette solution est beaucoup plus « optimisé » et beaucoup plus facile à lire.

Comment créer une méthode ?

C'est très simple, pour cela étudions l'exemple précèdent :

```
public static int somme (int tab[]){
		int somme=0;

		for (int i=0 ;i<tab.length;i++){
			somme+=tab[i];
		}
		return somme;
	}
```

Tout d'abord, il faut garder à l'esprit qu'une méthode est un programme qui peut fonctionner pour n'importe quelle donnée correspondant au type rentré en paramètre. Par exemple dans notre cas, la méthode pourra fonctionner avec n'importe quel tableau quel que soit les valeurs rentré (du moment que ce sont des entier) ou sa taille.

Etudions la forme :

```
public static int somme (int tab[])
```

public static : A mettre dans chacune de vos méthodes, il existe d'autre possibilité pour ces deux mots mais vous verrez ça en l2.

int : il s'agit du type que vous voulais renvoyer. (Attention un type « void » ne retournera rien !!!!)

Somme : c'est le nom de votre méthode elle vous servira plus tard à l'appeler.

(int tab[]) : il s'agit du type de donnée que vous voulez traiter, dans notre exemple un tableau d'entier.

Appel de la méthode : nomdelamethode(donnée_à_traiter)

Pensé à stocker le résultat de la méthode si vous voulez le réutiliser par la suite.

Pour ce faire :

Type nomvar=nomdelamethode(donnée_à_traiter) ;

Dans notre exemple :

```
int s1=somme(tab1);
```

 Pour un type « void » il n'y a rien à stocker !!!

IV-Fonction utile pour les chaines de caractères

-**int str.length()** : retourne la longueur de la chaîne de caractères str.

-**char str.charAt(int index)** : retourne le caractère de la chaîne str à l'indice index.

-**String str.substring(int indice)** : retourne une nouvelle chaîne qui commence à l'indice indice de la chaîne str.

-**String str.substring(int indiceDebut, int indiceFin)** : retourne une nouvelle chaîne comprise entre les indices indiceDebut et indiceFin de la chaîne str. Le caractère d'indice indiceFin n'est pas inclus dans la chaîne retournée.

-**int str.indexOf(char c)** : retourne l'indice de la 1ère occurrence du caractère c dans la chaîne str, sinon -1.

-**int str.lastIndexOf(char c)** : retourne l'indice de la dernière occurrence du caractère c dans la chaîne str, sinon -1.

-**int str1.indexOf(String str2)** : retourne l'indice de la 1ère occurrence de la sous-chaîne str2 dans la chaîne str1 en effectuant une recherche de gauche à droite. Si str2 n'est pas incluse dans str1, la méthode retourne -1.

-**int str1.lastIndexOf(String str2)** : retourne l'indice de la dernière occurrence de la sous-chaîne str2 dans la chaîne str1 en effectuant une recherche de gauche à droite. Si str2 n'est pas incluse dans str1, la méthode retourne -1.

-**int str1.indexOf(String str2, int indice)** : retourne la position de la 1ère occurrence de la sous-chaîne str2 dans la chaîne str1, en effectuant une recherche de gauche à droite à partir de l'indice indice. Si str2 n'est pas présente dans str1, à partir de l'indice indice, la méthode retourne -1.

-**boolean str1.startsWith(String str2)** : retourne true si la chaîne str1 débute par la sous-chaîne str2, sinon false.

-**boolean str1.endsWith(String str2)** : retourne true si la chaîne str1 se termine par la sous-chaîne str2, sinon false.

-**boolean str1.equals(String str2)** : compare si les deux chaînes de caractères str1 et str2 sont identiques. Si les deux chaînes sont identiques, la méthode retourne true, sinon elle retourne false.

-**String str1.concat(String str2)** : retourne une nouvelle chaine constituée de la chaîne str2 concaténée à la fin de la chaîne str1.

-**int str1.compareTo(String str2)** : compare alphabétiquement les deux chaines str1 et str2. Retourne une valeur <0 si la chaîne str1 est alphabétiquement avant str2 retourne 0 si la chaîne str1 est identique à str2 retourne une valeur >0 si la chaîne str1 est alphabétiquement après str2.

-**String [] str1.split(String str2)** : renvoie un tableau de chaines où chacune correspond à une sous-chaîne de str1 délimitée (séparée) par str2.

-**char [] str.toCharArray()** : retourne un tableau de caractères où chaque élément correspond à un caractère de la chaîne str.

-**String valueOf(Object obj)** : retourne la chaîne de caractères représentant l'objet obj. Ce dernier peur être d'un type de base (int, double, boolean, …) ou un objet tableau char [] ou String.

V-Récursivité

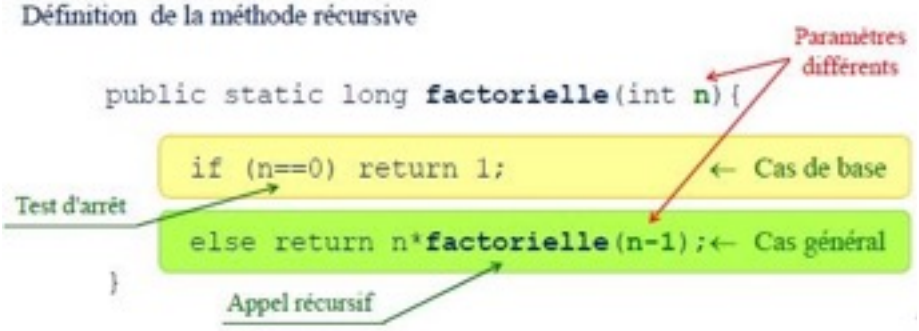
```
static  type recursiveM(type parametres);{
// declaration des variables locales

  if TEST_ARRET {
     …
     instructions du point d'arrêt      } pas d'appel récursif
     …
  }
  else {
     …                                  } présence
     recursiveM(nouveaux_parametres);   } d'appel(s)
     …                                  } récursif(s)
  }
}
```

Exemple de la factorielle :

Définition de la forme récurrente

```
0! = 1            ← Cas de base
n! = n.(n-1)!     ← Cas général
```

Définition de la méthode récursive



VI-Gestion des fichiers

Dernier point abordé en L1, nous allons séparer ce dernier en deux parties. Dans un premier temps, nous verrons la gestion des fichiers de données binaires puis nous terminerons par la gestion des fichiers textes.

Commençons par l'écriture dans des fichiers de données binaires. Prenons cet exemple :

```
DataOutputStream sortie = null;

try{

FileOutputStream f = new FileOutputStream(''monFichier'');
BufferedOutputStream buffer = new BufferedOutputStream(f);
sortie = new DataOutputStream(buffer);

sortie.writeInt(15);
sortie.writeInt(4);
sortie.writeInt(8);

}
catch(IOException e){

System.out.println(''Erreur fichier'');
}

finally{

sortie.close() ;
```

```
}
```

FileOutputStream f = new FileOutputStream(‘’monFichier‘’) : c

BufferedOutputStream buffer = new BufferedOutputStream(f) : Tampon créer afin d’optimiser les opérations sur le fichier.

sortie = new DataOutputStream(buffer) : Appel de la classe permettant d’écrire sur le fichier.

sortie.writeInt(15) : Ecriture d’un entier sur le fichier. (Remplacer .writeInt par la fonction d’écriture correspondant au type que vous voulez traiter)

Pour la lecture d’un fichier binaire c’est le même principe :

```
DataInputStream entree = null;

try{

FileInputStream f = new FileInputStream(‘’monFichier‘’);
BufferedInputStream buffer = new BufferedInputStream(f);
entree = new DataInputStream(buffer);
boolean eof = false;

while(!eof){

try{
n = entree.readInt();
}

catch(EOFException e){
eof = true;
}

if(!eof) System.out.println(n);
}

}
catch(IOException e){

System.out.println(‘’Erreur fichier‘’);
}

finally{
entree.close() ;
}
```

FileInputStream f = new FileInputStream(‘’monFichier‘’) : Ouverture du fichier binaire monFichier en lecture .

BufferedInputStream buffer = new BufferedInputStream(f) : Tampon créer afin d’optimiser les opérations sur le fichier.

entree = new DataInputStream(buffer) : Appel de la classe permettant de lire sur le fichier.

Principe de parcourt en lecture : Tant que eof(booleen) est vrai on parcourt le fichier binaire.

n = entree.readInt() : On stocke chaque donnée dans une variable que l’on affiche ensuite (Remplacer .readInt par la fonction d’écriture correspondant au type que vous voulez traiter).

Passons aux fichiers textes (même principe que les fichiers binaires) :

Méthode d’écriture :

```
PrintWriter sortie = null;
try{
FileWriter f = new FileWriter(‘’monFichier‘’);
BufferedWriter buffer = new BufferedWriter(f);
sortie = new PrintWriter(buffer);
sortie.println(‘’1ère ligne‘’);
sortie.println(‘’2ème ligne‘’);
}
catch(IOException e){
System.out.println(‘’Erreur fichier‘’);
}
finally{
sortie.close()
}
```

Méthode de lecture :

```
BufferedReader entree = null;
 try{
FileReader f = new FileReader(‘’monFichier‘’);
entree = new BufferedReader(f);
String ligne = ‘’ ‘’;
 while(ligne != null){
 ligne = entree.readLine();
 if(ligne != null)
System.out.println(ligne);
}
```

```
} catch(IOException e){
System.out.println(‘’Erreur fichier‘’);
} finally{
entree.close()
}
```