DEUST
Systèmes d'information et Réseaux,
Gestion et réalisation

1ère année

Langage à objets : JAVA (1)

2021-2022

Introduction

Langages compilés et interprétés

Langage compilé

Lorsqu'un langage informatique est destiné à être **compilé**, un **compilateur** dédié au langage examine le code source.

Ce compilateur contrôle si le code source vérifie la syntaxe du langage utilisé. Si c'est le cas, le compilateur génère un fichier en langage machine, dit "**code objet**" exécutable sur tout ordinateur **du même type** que celui sur lequel on a effectué la compilation.

Exemples de langages compilés : Pascal, Cobol, C, C++

Avantage des langages compilés

Le code objet s'exécute très **rapidement**, ce qui est indispensable pour les applications économiques ou industrielles.

Inconvénients des langages compilés

La mise au point des programmes est délicate (en cas de plantage pendant l'exécution, où est l'erreur ?)
Le compilateur dépend du **type d'ordinateur** sur lequel on travaille.
Le code objet est **dédié** au type de machine sur lequel on a compilé.

Langage interprété

Lorsqu'un langage est **interprété**, l'**interpréteur** ne traduit pas globalement le fichier source en un fichier exécutable.

Il lit au contraire le fichier source ligne à ligne, vérifie chaque instruction et l'exécute si elle est correcte.

Exemples de langages interprétés : HTML, JavaScript, Perl, PHP.

Avantages des langages interprétés

L'interpréteur s'arrête en cas d'erreur sur la ligne erronée ; le "débuguage" est donc facilité.

Le type de machine qui interprète le fichier n'est pas forcément le même que celui sur lequel on a tapé le fichier source. Il suffit de disposer de l'interpréteur adéquat sur la machine destinataire. Un programme écrit en langage interprété est donc **portable**.

Inconvénient des langages interprétés

La vérification ligne à ligne du code source cause une certaine **lenteur** à l'exécution.

Java, compilé et interprété

Java est un langage développé par la société SUN en 1995 pour assurer la portabilité des programmes sur une large gamme de machines.

Sur l'ordinateur sur lequel on écrit un programme Java, le code source est transformé par le **compilateur javac** en un « **pseudo-code »** non exécutable, qui est **le même** quel que soit le type d'ordinateur sur lequel on travaille.

Transféré sur un autre type d'ordinateur, le pseudo-code est **interprété** par **l'interpréteur java**, qui ne vérifie plus la syntaxe des instructions, mais exécute seulement le pseudo-code.

A noter

Le pseudo-code est **indépendant** du type d'ordinateur sur lequel il a été créé et du type d'ordinateur sur lequel il va être exécuté. Il est donc **portable**.

Pour **écrire** du java, il suffit de disposer du **compilateur javac** adapté à la machine sur laquelle on travaille ; ce compilateur génère du pseudo-code (le même quelle que soit la machine).

Pour **exécuter** du java, il suffit de disposer de **l'interpréteur java** adapté à la machine sur laquelle on travaille. Cet interpréteur lit et exécute le pseudo-code.

Lors de la compilation, le code des bibliothèques utilisées n'est **pas intégré** dans le pseudo-code ; seul le bon usage de ces bibliothèques est vérifié à la compilation.

Mais le code des bibliothèques utilisées est chargé à l'exécution. Le pseudo-code est donc de **faible taille**, et circule rapidement sur les réseaux.

Conséquence : java est un langage idéal pour l'**Internet**.

Ce cours permet d'étudier les applications non graphiques. Les applications graphiques seront vues en deuxième année.

TD1 - Invite de commandes MS-DOS

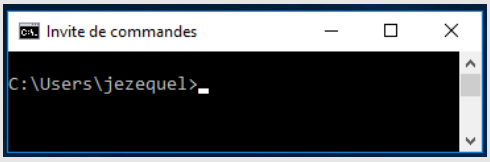
Récupération de fichiers sur le serveur

Recopiez sur le bureau le dossier JAVA qui se trouve sur le serveur.
A la fin de la séance, vous recopierez sur votre clé USB l'ensemble du dossier.

Accès à l'invite de commandes MS-DOS

Le MS-DOS est le premier système d'exploitation de Microsoft, où l'on s'exprime en tapant des lignes de commande (analogue aux shells Unix). On en a besoin lorsqu'on souhaite réaliser des opérations n'exploitant pas Windows.
Le MS-DOS est accessible en tapant *cmd* dans la barre de recherchede Windows (en bas à gauche) ou par une icône sur le bureau**.** Dans la fenêtre ainsi ouverte appelée l'invite de commandes, l'interpréteur MS-DOS attend les commandes de l'utilisateur.

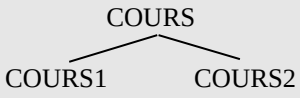
Exemple d'invite de commandes :



Ici le prompt initial est C:\Users\jezequel>. Il sera bien sûr différent sur votre ordinateur.

Arborescence des dossiers

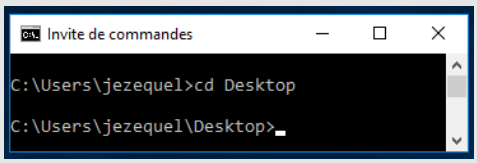
Dans cet exemple, nous avons un dossier COURS et deux sous-dossiers COURS1 et COURS2. Chaque dossier peut éventuellement contenir des fichiers (textes, images, programmes, …).



Commande pour aller dans le dossier COURS : **cd COURS**
Commande pour aller ensuite dans le sous-dossier COURS1 : **cd COURS1**
Commande pour remonter au niveau supérieur (ici on va se retrouver dans le dossier COURS) : **cd ..**

Initiation au MS-DOS : commandes MS-DOS courantes

1) Aller sur le bureau, il suffit de taper :
cd Desktop et valider



cd est l'abbréviation de « change directory » qui permet de changer le dossier courant. Remarquer le changement de prompt.

Aller ensuite dans le dossier JAVA, il suffit de taper :
cd java et valider
Remarquer à nouveau le changement de prompt.

2) Regarder ce qui se trouve dans le dossier JAVA. Il s'agit de la commande **dir** :
dir liste tous les fichiers et les sous-dossiers du dossier courant
dir e*.* liste les fichiers et dossiers commençant par la lettre e.

3) Aller dans le dossier UTILTRUC : comme on est déjà dans JAVA, il suffit de taper :
cd utiltruc
Remarquer toujours le changement de prompt.

4) Regarder ce qui se trouve dans le dossier UTILTRUC :
dir

5) Revenir dans JAVA :
cd ..

6) Créer un dossier TEST dans JAVA: il s'agit de la commande **md** (**m**ake **d**irectory) donc :
md test

7) Recopier le fichier emploi.doc de JAVA dans le dossier TEST: il s'agit de la commande **copy**.
copy emploi.doc test

8) Vérifier le contenu de TEST:
dir test

9) Aller dans TEST et renommer emploi.doc en truc.doc : il s'agit de la commande **rename**.
cd test
rename emploi.doc truc.doc

10) Supprimer le fichier truc.doc. il s'agit de la commande **del**.
del truc.doc

11) Revenir dans JAVA
cd ..

12) Supprimer le dossier TEST. Il s'agit de la commande **rd** (**r**emove **d**irectory).
rd test

Les commandes MS-DOS à connaître

Indiquez pour chaque action la commande MS-DOS correspondante.

1. afficher le contenu du dossier courant : ……..……..
2. aller dans le répertoire TEST : ……..……

3. revenir au dossier « parent » : ……..…….
4. créer le dossier EXEMPLE (make directory) : ……..……...
5. effacer le fichier Ex.java : ……..……...
6. effacer le dossier EXEMPLE (remove directory) : ……..……
7. renommer le fichier Projet.java en Appli.java : ……..……...
8. copier le fichier Ex1.java en Ex2.java : ……..……...

Eléments de programmation Java

Notion de variable

La notion de **variable** en informatique est assez proche de la notion mathématique.

En mathématiques, x = 5 veut dire que la variable x **vaut** 5.

En informatique, x est le **contenu** d'un **emplacement-mémoire** dont la taille (nombre de bits qui composent cet emplacement) dépend de ce qu'on veut y stocker (un caractère prend moins de place qu'un entier, un entier prend moins de place qu'un réel…). Le contenu de cet emplacement-mémoire peut constamment être **changé**.

On précise donc (syntaxe java) :

```
int x ;          (réservation de place-mémoire pour un entier)
x = 5 ;          (on y met la valeur 5)
x = 7 ;          (on remplace 5 par 7 dans le même emplacement mémoire)
float y ;        (réservation de place-mémoire pour un réel)
y = 3.5 ;        (on y met la valeur 3.5)
```

Ces lignes peuvent être résumées en :

```
int x = 5;x = 7;  (réservation de place-mémoire pour un entier et on y met la valeur 5 puis 7)
float y = 3.5 ;   (réservation de place-mémoire pour un réel et on y met la valeur 3.5)
```

Affectation

Les instructions `x = 5 ; x = 7 ; y = 3.5 ;` donnent des valeurs aux emplacements-mémoire x et y. Ces instructions sont appelées des **affectations**.

Il serait plus correct d'écrire respectivement : `x` ← `5` (x reçoit 5), `x` ← `7` (x reçoit 7) et `y` ← `3.5` (y reçoit 3.5) mais la plupart des langages de programmation (C, C++, JavaScript, Java…) utilisent la notation `a = b` au lieu de `a` ← `b`.

L'écriture **`a = b`** correspond donc en fait à **`a` ← `b`** et doit se lire **a "reçoit" la valeur de b**.

On peut affecter à une variable x :

une valeur : `x = 5 ;`

la valeur d'une autre variable : `y = 3 ; x = y ;`

le résultat du calcul d'une expression : `y = 3 ; z = 4 ; x = y + z ;`

Types de données primitifs en Java

En Java, il y a 8 types de données primitifs. Ce sont les types (en **gras** les plus utilisés) :

boolean	(true et false)
char	(caractère),
byte	(petit entier stocké sur 8 bits),
short	(entier court stocké sur 16 bits),
int	(entier "normal" stocké sur 32 bits),

long	(entier long stocké sur 64 bits),
float	(flottant simple stocké sur 32 bits),
double	(flottant double précision stocké sur 64 bits).

L'instruction `boolean fini = false ;` définit une variable booléenne, appelée fini et de valeur false.

L'instruction `char c = 'c' ;` définit une variable de type caractère, appelée c et de valeur 'c'.

L'instruction `int i = 254256 ;` définit une variable de type entier, appelée i, stockée sur 32 bits et de valeur 254256.

L'instruction `long l = 25425645987 ;` définit une variable de type entier long, appelée l, stockée sur 64 bits et de valeur 25425645987.

L'instruction `double f = 156.256 ;` définit une variable de type flottant double précision, appelée f, stockée sur 64 bits et de valeur 156.256.

Attention au type des variables !

On considère le résultat d'une opération arithmétique entre deux variables a et b.

a et b entiers	→	résultat entier
a ou b flottants	→	résultat flottant
a et b flottants	→	résultat flottant

Exemples :

```
int a=1, b=2;
int i=a/b ;              Que vaut i ?

float a=1.,b=2.;
float x=a/b ;            Que vaut x ?
```

Opérateurs de cast

Pour transformer une variable d'un type primitif en un autre, on utilise un opérateur de "**cast**" qui force la **conversion**. Selon le cas, on peut perdre de la précision.

Exemples :

```
double f = 156.256 ; int j = (int) f ;          Que vaut j ?
int i=3; double x=1/i ; double y=1/(double)i ;  Que vaut x ?    Que vaut y ?
```

Application et méthode main

Soit le programme suivant, sauvegardé sous le nom **Programme.java** dans le dossier PROGRAMME :

```
public class Programme {
      public static void main(String[] args) {
           int a = 25 ;
           int b = 76 ;
           System.out.println("a = " + a);
           System.out.println("b = " + b);
           }
      }
```

Tout programme java est nécessairement contenu dans une **classe**. Lorsque le programme ne comporte qu'une seule classe, celle-ci est déclarée **public** afin d'être accessible à tout utilisateur.

Le programme proprement dit figure dans une méthode appelée **main** nécessairement déclarée sous la forme `public static void main(String[] args)`.

Remarques

`String[] args` permet passer des informations à la méthode main, sous la forme d'un tableau de chaînes de caractères de type `String[]`. On n'utilise cette possibilité que lorsqu'on veut donner des informations au programme principal, dès son lancement.

`System.out.println(chaîne)` permet de faire afficher à l'écran une chaîne de caractères simple ou composée. Pour construire une chaîne composée à partir de plusieurs morceaux, on concatène ces morceaux en utilisant le signe +

TD2 - Compilation, exécution, modification d'une application

1) Ouvrir avec un éditeur le fichier **Programme.java** qui se trouve dans le dossier PROGRAMME. Comme tous les fichiers source java, c'est un fichier sauvegardé en format texte uniquement.

2) Dans la fenêtre MS-DOS, taper la commande **PATH**. Vérifier que le PATH contient le chemin menant à un compilateur java : JDK….\BIN.

   Rôle du PATH ?

3) Se placer dans le dossier PROGRAMME. Taper la commande **javac Programme.java**. Si la compilation se passe bien, un fichier de nom **Programme.class** est généré par le compilateur (vérifier avec **dir**). Sinon, il faut corriger la commande et recompiler.

4) Toujours dans la fenêtre MS-DOS, faire exécuter le fichier par la commande **java Programme** (sans l'extension .class). L'interpréteur java lance automatiquement la méthode main.

5) Ajouter au main de la classe Programme les lignes permettant de déclarer deux entiers c et d, de calculer c=a+b et d=a*b, puis de faire afficher les valeurs de c et de d. Compiler et faire exécuter le fichier ainsi modifié.

6) On voudrait à présent que a, b, c et d puissent être des points à coordonnées réelles (de type primitif double) et non plus uniquement entières. Modifier la classe Programme en conséquence, en prenant cette fois a=4.56 et b=-8.65. Compiler le fichier Programme.java ainsi modifié et le faire exécuter.

7) On veut faire afficher en fin de programme « valeur entiere de d : » puis la valeur entière de d. Utiliser un opérateur de cast pour obtenir ce résultat.

8) Bonus : créer un dossier AGRO et y écrire un programme qui, à partir de la longueur et de la largeur d'un champ, en affiche le périmètre et la surface.

Méthodes de classe

Soit le programme suivant, enregistré sous le nom **Programme0.java** dans le dossier PROGRAMME0 :

```
public class Programme0 {
      static void affiche(int n) {
            System.out.println(n);
            }
      public static void main(String[] args) {
            int a = 25 ; int b = 76 ;
            affiche(a) ;affiche(b);
            System.out.print("somme de a et b : ") ;affiche(a+b) ;
            }
      }
```

`static void affiche(int n) {...}` est appelée une **méthode de classe** de la classe Programme0. Elle permet d'afficher un entier n passé en paramètre.

`static` indique que c'est une méthode de **classe** (on verra qu'il y a d'autres types de méthodes)
`void` indique que la méthode ne renvoie pas de résultat
`affiche` est le nom de la méthode
`int n` est l'argument de la méthode.

Les tests

Les tests permettent de faire des traitements différenciés en fonction du contexte.

Exemple 1 : on cherche le plus petit de deux nombres a et b.

Si a < b, alors le plus petit est a ; sinon le plus petit est b.
En java, le test « if » permet de répondre à la question.

Il s'écrit :

```
if (condition) {
      instructions;
      }
else  {
      instructions;
      }
```

On aura donc :

```
if (a < b) {
      return a;
      }
else  {
      return b;
      }
```

Exemple 2 : la variable entière num contient un nombre entre 1 et 12. On veut récupérer le nom du mois dans une variable mois.

En java, le test « switch » permet de répondre à la question.

Il s'écrit :

```
switch (variable) {
 case 1: instructions;
 case 2: instructions;
 ...
 default : instructions;
 }
```

On aura donc :

```
switch(num) {
 case 1 : mois="janvier";
 case 2 : mois="février";
 ...
 case 12 : mois="décembre";
}
```

Le cas `default` sert si l'on veut prévoir un traitement par défaut.

Opérateurs relationnels pour exprimer une condition :

`if (a>b)`	Si a est strictement supérieur à b
`if (a>=b)`	Si a est supérieur ou égal à b
`if (a<b)`	Si a est strictement inférieur à b
`if (a<=b)`	Si a est inférieur ou égal à b
`if (a==b)`	Si a et b ont la même valeur
`if (a !=b)`	Si a et b n'ont pas la même valeur

Opérateurs logiques

	Et : `&&`	Ou : `||`
Exemples :	`if ((a==2) && (b==4)) {...}`	`if ((a==2)) || (b==4)) {...}`

TD3 - Tests

Soit le programme suivant, enregistré sous le nom **Programme1.java** dans le dossier PROGRAMME1 :

```
public class Programme1 {
     static int min(int a, int b) {
           if (a < b) { return a ; }
                 else { return b; }
           }
     public static void main(String[] args) {
           int a = 25 ;
           int b = 76 ;
```

```
            System.out.println("a = " + a);
            System.out.println("b = " + b);
            System.out.println("minimum de a et b : " + min(a,b));
            }
      }
```

1) Aller dans PROGRAMME1, taper la commande **javac Programme1.java** puis **java Programme1**

2) Ouvrir avec un éditeur le fichier **Programme1.java** et ajouter à la classe une méthode
`static int max(int a, int b) {...}`
qui renvoie le plus grand des nombres a et b. Ajouter au main la ligne nécessaire pour faire afficher "`maximum de a et b : ...`" qui renvoie le plus grand des nombres a et b. Compiler et exécuter le programme.

3) Ajouter à la classe une méthode `static int somme(int a, int b) {...}` qui renvoie la somme des nombres a et b. Ajouter au main la ligne nécessaire pour faire afficher "`la somme de a et b est...`" qui renvoie la somme des nombres a et b. Compiler et exécuter le programme.

Les tableaux

Les tableaux sont des collections de variables **de même type**, en **nombre fixé**. On doit les déclarer, puis faire l'allocation de mémoire nécessaire :

```
int [] t ;              // déclaration d'un tableau d'entiers t
t = new int[20] ;       // création de t en mémoire (20 entiers)

char [] tab ;           // déclaration d'un tableau de caractères tab
tab = new char[40] ;    // création de tab en mémoire (40 caractères)

String [] t1 ;          // déclaration d'un tableau t1 de chaînes
t1 = new String[10] ;   // création de t1 en mémoire (10 chaînes)
```

On accède à l'élément d'indice i de t par **`t[i]`**, **les indices commençant à 0**.
Une fois qu'un tableau est créé, **sa taille ne peut plus être changée**.

Utilisation de tableau
Exemple : soit le programme suivant, enregistré sous le nom **Tableau.java** dans le dossier TABLEAU. Il comporte deux méthodes de classe : la méthode **main** et la méthode **affiche**. Le compiler et l'exécuter.

```
public class Tableau {

      static void affiche(String [] tab) {
            System.out.println("Les mois de l'annee sont : ");
            System.out.println(tab[0]);
            System.out.println(tab[1]);
            System.out.println(tab[2]);
            System.out.println(tab[3]);
            System.out.println(tab[4]);
            System.out.println(tab[5]);
            System.out.println(tab[6]);
            System.out.println(tab[7]);
            System.out.println(tab[8]);
            System.out.println(tab[9]);
            System.out.println(tab[10]);
            System.out.println(tab[11]);
            }
      public static void main(String[] args) {
            String[] t ;
            t = new String[12] ;
            t[0]="janvier"; t[1]="fevrier"; t[2]="mars"; t[3]="avril";
            t[4]="mai"; t[5]="juin"; t[6]="juillet"; t[7]="aout";
```

```
            t[8]="septembre"; t[9]="octobre"; t[10]="novembre";
            t[11]="decembre";
            affiche(t);
            }
      }
```

Les boucles

La boucle for

La boucle **for** permet de répéter un certain nombre de fois des instructions. Elle s'écrit :
`for (initialisation; test de continuation; incrémentation) {instructions;}`
L'initialisation permet de préciser la variable qui contrôle la boucle et sa valeur initiale : `int i = 0`
Le test de continuation dit quel est le test qui doit être vérifié pour que la boucle continue : `i < 20`
L'incrémentation précise comment évolue la variable de contrôle : `i++` veut dire `i=i+1` (on passe de i à i+1), mais on pourrait aussi avoir `i=i+2` (pour sauter un élément sur deux) ou `i=i+10` (pour ne traiter qu'un élément sur 10), ou toute autre combinaison.

Le fichier **Tableau1.java** enregistré dans le dossier TABLEAU1 permet de réduire l'écriture du programme Tableau précédent. Le compiler et l'exécuter.

```
public class Tableau1 {
     public static void main(String[] args) {
            String[] t ; t = new String[12] ;
            t[0]="janvier"; t[1]="fevrier"; t[2]="mars";t[3]="avril";
            t[4]="mai"; t[5]="juin"; t[6]="juillet"; t[7]="aout";
            t[8]="septembre"; t[9]="octobre"; t[10]="novembre";
            t[11]="decembre";
            System.out.println("Les mois de l'annee sont : ");
            for (int i = 0 ; i < 12; i++) {
                  System.out.println(t[i]);
                  }
            }
     }
```

Dans le programme **Tableau2.java** du dossier TABLEAU2, on utilise **deux boucles for** : une pour initialiser le tableau, l'autre pour faire afficher ses éléments :

```
public class Tableau2 {
     public static void main(String[] args) {
            double[] t = new double[20] ;
            for (int i = 0 ; i < 20 ; i++) {
                  t[i] = i ;
                  }
            for (int i = 0; i < 20 ; i++) {
                  System.out.print(t[i] + " * ");
                  }
            }
     }
```

La même variable **i** sert à contrôler les deux boucles, mais comme elle est déclarée **dans** l'instruction for, elle doit être déclarée chaque fois. **Entre les deux boucles for, la variable i n'existe pas.**
Par contre, en déclarant la variable i dans le main, la variable i est connue de tout le programme principal :

```
public class Tableau2 {
     public static void main(String[] args) {
            double[] t = new double[20] ; int i ;
            for (i = 0 ; i < 20 ; i++) {
                  t[i] = i ;
                  }
```

```
            for (i = 0; i < 20 ; i++) {
                  System.out.print(t[i] + " * ");
                  }
            }
      }
```

TD4 : Tableaux et boucles for

1) Ouvrir le fichier **Tableau2.java** avec un éditeur, compiler le fichier et le faire exécuter.

2) On décide à présent d'initialiser le tableau t avec les multiples de 3, soit 3 6 9 12 ... Modifier le main en conséquence, compiler et exécuter

3) Modifier la classe Tableau2 de la manière suivante et compléter la méthode **affiche** pour parvenir au même affichage que précédemment :

```
public class Tableau2 {
      static void affiche(double[] tableau, int n) {
            ...
            }
      public static void main(String[] args) {
            double[] t ;
            t = new double[20] ;
            for (int i = 0 ; i < 20 ; i++) {
                  t[i] = i ;
                  }
            affiche(t, 20) ;
            }
      }
```

Cette nouvelle version permet de disposer d'une méthode d'affichage qui peut être appelée avec **différents tableaux de tailles variables**.

4) Tester en ajoutant les lignes suivantes au main :

```
            System.out.println() ;
            double[] t1 ;
            t1 = new double[10] ;
            for (int i = 0 ; i < 10 ; i++) {
                  t1[i] = 25*i ;
                  }
            affiche(t1, 10) ;
```

Compiler et exécuter.

TD5 : un placement fructueux ?

Compléter le fichier **LivretA.java** du dossier LivretA afin qu'il calcule et affiche le montant atteint chaque année par un placement à taux fixe. Le programme pourra par exemple afficher le montant atteint chaque année pendant 10 ans par un montant initial de 1000 € placé à 3%.

La boucle while

La boucle while permet de répéter des instructions tant qu'un test est vérifié. Ces instructions doivent modifier **au moins** une variable intervenant dans le test, pour que le while ne boucle pas indéfiniment.

Exemple : afficher les éléments du tableau de nombres t tant que ces éléments sont inférieurs à 20.

La syntaxe est :

```
while (test) {
    instructions ;
    }
```

On écrira donc :

```
int i = 0 ;
while (t[i] < 20) {
      System.out.print(t[i]) ;
      i++ ;
      }
```

Dans cet exemple, i est modifié dans les instructions, donc t[i] correspond à chaque passage à un nouvel élément du tableau t.

Attention cependant à ne pas dépasser la taille du tableau ! Si le tableau est de taille n, il faut écrire :

```
int i = 0 ;
while ((i < n) && (t[i] < 20)) {    // && est la notation pour ET
      System.out.print(t[i]) ;
      i++ ;
      }
```

La boucle do … while

La boucle do … while permet de faire exécuter une première fois des instructions puis de répéter ces instructions tant qu'un test est vérifié. Ces instructions doivent modifier **au moins** une variable intervenant dans le test, pour que le do … while ne boucle pas indéfiniment.

Exemple : afficher le premier élément du tableau t, puis les éléments suivants du tableau tant que ces éléments sont inférieurs au premier (vérifier que l'on ne dépasse pas la fin du tableau).

La syntaxe :

```
do    {
      instructions ;
      }
while (test);
```

On écrira donc :

```
int i = 0 ;
do {
    System.out.print(t[i]);
    i++ ;
    }
while ((i<n)&&(t[i]<t[0]));
```

TD6 - Tableaux et boucles while ou do… while

1) Ouvrir le fichier **Tableau3.java** du dossier TABLEAU3 avec un éditeur, compiler le fichier et le faire exécuter. Ce programme fait appel à un **générateur de nombres aléatoires** appelé r. Il génère ici 20 nombres entiers compris entre 0 (inclus) et 50 (exclus). La méthode de classe **affiche** fait afficher les éléments d'un tableau tab de taille nb, tab et nb étant passés en paramètres.

```
public class Tableau3 {
     static void affiche(double[] tab, int nb) {
           for (int i = 0; i < nb; i++) {
                 System.out.print(tab[i] + " * ");
                 }
           }
     public static void main(String[] args) {
           int n=20 ;
           double[] t = new double[n] ;
           java.util.Random r = new java.util.Random();
           for (int i = 0 ; i < n ; i++) {
                 t[i] = r.nextInt(50) ;
                 }
           affiche(t, n);
           }
     }
```

2) Transformer la méthode affiche pour qu'elle affiche les éléments du tableau tab **tant que ceux-ci sont inférieurs ou égaux à 40** (utiliser une boucle while).

3) Faire exécuter le programme plusieurs fois de suite. Remplacer ensuite 40 par 50 et vérifier que le programme affiche bien tous les éléments du tableau.

4) Modifier la méthode affiche pour qu'elle affiche le premier élément du tableau tab, puis les éléments suivants **tant que ceux-ci sont inférieurs ou égaux au premier élément** (utiliser une boucle do … while).

Les sauts

Le break

L'instruction **break** permet de sortir d'un test ou d'une boucle en reprenant l'exécution directement **après** le test ou la boucle (on saute donc tous les cas suivants).

Exemple : afficher les éléments du tableau t tant qu'ils sont inférieurs à 40

```
for (int i = 0 ; i < n ; i++) {
   if (t[i]<40) {
      System.out.print(t[i]+" * ");
      }
   else {
      break ;
      }
   }
```

C'est équivalent à la boucle while ci-dessous :

```
int i = 0 ;
while ((i<n) && (t[i] < 40)) {
      System.out.print(t[i]+" * ") ;
      i++ ;
      }
```

Le continue

L'instruction **continue** permet de sortir d'un pas d'itération en reprenant l'exécution à l'itération suivante. On arrête donc les instructions de l'itération en cours pour passer à l'itération suivante.

Exemple : afficher les éléments du tableau t différents de 20
Deux versions équivalentes, selon que l'on teste si t[i] est égal à 20 ou est différent de 20.
L'opérateur **d'égalité** entre nombres est == (double égal) tandis que l'opérateur **d'inégalité** est !=

```
for (int i = 0 ; i < n ; i++) {
   if (t[i] == 20) {
      continue ;
      }
   else {
      System.out.print(t[i]+" * ");
      }
   }
```

```
for (int i = 0 ; i < n ; i++) {
   if (t[i] != 20) {
      System.out.print(t[i]+ " * ");
      }
   else {
      continue ;
      }
   }
```

Le return

L'instruction **return** permet la sortie d'une **méthode** sans nécessairement attendre la fin de celle-ci. Cette instruction ne peut donc pas figurer directement dans le main.

Exemple : afficher les éléments d'un tableau tant qu'ils sont inférieurs à 30

```
static void affiche(double[] tab, int nb) {
      for (int i = 0; i < nb ; i++) {
            if (tab[i] < 30) {
                  System.out.print(tab[i] + " * ");
                  }
            else {
                  return ;
                  }
            }
      }
```

TD7 - Tableaux et sauts

1) Ouvrir le fichier **Tableau4.java** du dossier TABLEAU4 avec un éditeur, compiler le fichier et le faire exécuter.

2) Modifier la méthode affiche pour qu'elle affiche les éléments du tableau jusqu'au moment où l'élément rencontré vaut 25.

3) Ajouter ensuite un argument à affiche en écrivant :
`static void affiche(double[] tab, int nb, int nombre)` et modifier la méthode pour qu'elle affiche les éléments du tableau **jusqu'au moment où l'élément rencontré vaut** « nombre ».
Modifier le main en conséquence en remplaçant `affiche (t, n);` par : `affiche (t, n, 30);` et tester.

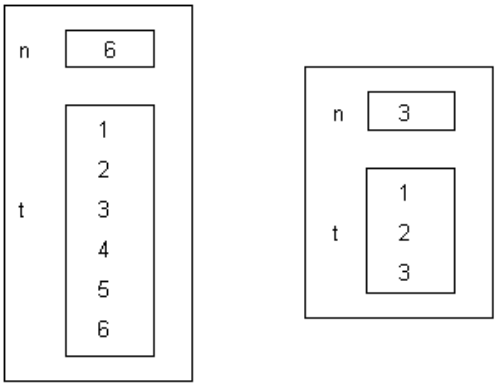
Java, langage à objets

Un programme java peut ne contenir qu'une seule classe, comme on l'a vu jusqu'à présent. Cette classe, qui contient alors la méthode main, constitue à elle toute seule toute l'application.

Classes et instances

Mais java est avant tout un "**langage à objets**", ce qui veut dire qu'une **classe** peut aussi être un "moule" à partir duquel on peut **fabriquer des objets** qu'on appelle des **instances.**
On souhaite créer des « objets » comportant :
- un entier n qui représente la taille d'un tableau
- un tableau t de n entiers comportant les entiers 1, 2, 3, … n

Exemple de deux tels objets :



La classe (encore embryonnaire) correspondant à ces objets est la suivante :

```
class ObjetTableau {   // début de la classe
     int n ;           // deux attributs : n entier
     int[] t ;         // et t tableau d'entiers
     }                 // fin de la classe
```

Cette description signifie que **tout objet** de la classe ObjetTableau, **dès qu'il sera créé**, comportera automatiquement **deux variables** (qu'on appellera des **attributs**) **n** et **t**.

Une **classe** correspond donc à la description d'un ensemble d'objets ayant **une structure de données commune**. Les **objets** créés selon ce modèle sont appelés des **instances** de la classe.

Pour donner des valeurs aux attributs, donc remplir n et t, on doit ajouter dans la classe ObjetTableau une méthode spécifique appelée **constructeur** :

```
class ObjetTableau {
     int n ;
     int[] t ;
```

```
    ObjetTableau(int a) {   // le constructeur
          n = a ;
          t = new int[n] ;
          for (int i=0 ; i < n ; i++) {
                t[i] = i+1 ;
                }
          }                 // fin du constructeur
    }
```

Créer des objets de la classe ObjetTableau se fait **à l'extérieur**, dans une **autre classe** qui sert de programme principal :

```
public class ProgObjet {                                  // début de classe
    public static void main (String args[ ]) {
          ObjetTableau obj1 = new ObjetTableau(6);
          ObjetTableau obj2 = new ObjetTableau(3);
          }
    }                                                     // fin de classe
```

Ce programme crée bien les deux objets présentés page précédente, mais n'affiche rien. On ajoute donc à la première classe une méthode d'affichage pour un quelconque de ses objets et une méthode **`int inferieurs(int p)`** calculant le nombre d'éléments du tableau t inférieurs à p :

```
class ObjetTableau {
    int n ;
    int[] t ;
    ObjetTableau(int a) {   // le constructeur
          n = a;
          t = new int[n] ;
          for (int i=0;i<n;i++) {
                t[i] = i+1 ;
                }
          }                 // fin du constructeur
    void affiche() {
          System.out.println("affichage d'un objet");
          System.out.println("n vaut "+n);
          for (int i = 0; i < n; i++){
                System.out.print(t[i]+" * ");
                }
          System.out.println("");
          }
    int inferieurs(int p) {
          int compteur = 0 ;
          for (int i = 0; i < n; i++){
                if (t[i]<p) compteur++ ;
                }
          return compteur ;
          }
    }
```

La méthode **`affiche`** ne peut être exécutée que par une instance (**méthode d'instance**). Elle permet d'afficher un objet à l'écran, mais elle ne calcule **rien**, d'où son type de **valeur de retour** : **`void`**.
La méthode **`inferieurs`** ne peut être exécutée que par une instance (**méthode d'instance**). Elle renvoie le nombre d'éléments du tableau t inférieurs à un entier p donné, d'où son type de **valeur de retour** : **`int`**.
On modifie en conséquence le programme principal :

```
public class ProgObjet {
    public static void main (String args[ ]) {
          ObjetTableau obj1 = new ObjetTableau(6);
```

```
            obj1.affiche();
            System.out.print("Nb elements inferieurs a 3 : ");
            System.out.println(obj1.inferieurs(3)) ;
            ObjetTableau obj2 = new ObjetTableau(3);
            obj2.affiche();
            System.out.print("Nb elements inferieurs a 4 : ");
            System.out.println(obj2.inferieurs(4)) ;
            }
      }
```

Ce fichier comporte deux classes : la classe ObjetTableau et la classe ProgObjet. Il y aura donc deux fichiers générés : ObjetTableau.class et ProgObjet.class. Compiler le fichier **ProgObjet.java** dans le dossier PROGOBJET et faites-le exécuter.

Il s'agit d'une **application**, car il y a dans une des classes une méthode appelée **main**. La **classe principale** est celle qui comporte la méthode main ; elle doit impérativement être déclarée **public**. On l'appelle souvent aussi **classe maître** de l'application.

Le fichier **doit** porter le nom de cette classe maître, suivi de l'extension .java : **ProgObjet.java**

Attributs et méthodes d'instances

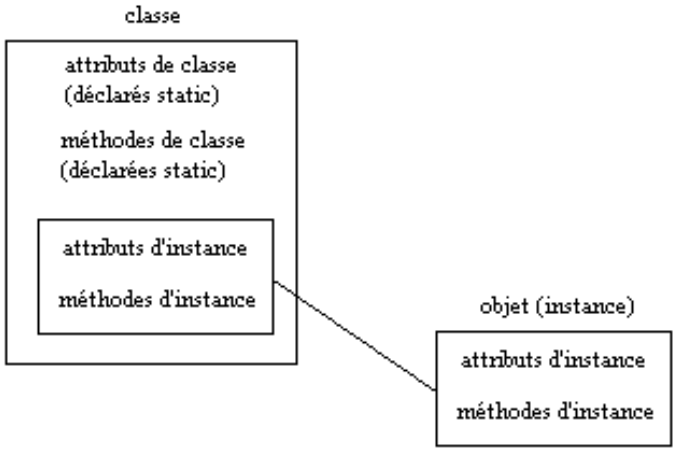
Une **classe** est un " modèle " permettant de construire des objets. Un objet est nécessairement créé à partir d'une classe, à l'aide d'un constructeur de cette classe ; on dit que l'objet est une **instance** de la classe.

Un objet comporte des **attributs d'instances** et dispose de **méthodes d'instances**, définis dans la classe.

Ainsi, dans la classe ObjetTableau,
n et t sont des **attributs d'instance**
la méthode affiche() est une **méthode d'instance**.

Attributs et méthodes de classes

Une classe peut aussi comporter des **attributs de classe** et des **méthodes de classe**. Ils sont signalés par le mot-clé **static** et existent dès le début de l'application, même si aucun objet de la classe n'a encore été créé. **Seule exception** : le **constructeur** est une méthode de classe, même s'il n'est pas précédé de static.




```
class ObjetTableau {
      static int nbobjets = 0;      // initialisation du compteur
      int n ; int[] t ;

      ObjetTableau(int a) {
            n = a;
            t = new int[n] ;
```

```
            for (int i=0;i<n;i++) {
                  t[i] = i+1 ;
                  }
            nbobjets++ ;            // incrémentation du compteur
            }
      void affiche() {
            System.out.println("affichage d'un objet");
            System.out.println("n vaut "+n);
            for (int i = 0; i < n; i++){
                  System.out.print(t[i]+" * ");
                  }
            System.out.println("");
            }
      static void nombreObjets() {  // méthode de classe
            System.out.println("Nombre d'objets : " + nbobjets) ;
            }
      }

public class ProgObjet1 {
      public static void main (String args[ ]) {
            ObjetTableau.nombreObjets() ;
            ObjetTableau obj1 = new ObjetTableau(6);
            obj1.affiche();
            ObjetTableau.nombreObjets() ;
            ObjetTableau obj2 = new ObjetTableau(3);
            obj2.affiche();
            ObjetTableau.nombreObjets() ;
            }
      }
```

TD8 - Méthodes de classe et d'instances

Se positionner dans le dossier PROGOBJET1. Compiler le fichier **ProgObjet1.java** puis le faire exécuter.
Noter dans le tableau suivant si la méthode appelée est une méthode de classe ou d'instance, puis le résultat de son exécution :

instruction	méthode : classe / instance	affichage
`ObjetTableau.nombreObjets();`		
`ObjetTableau obj1 = new ObjetTableau(6);`		
`obj1.affiche();`		
`ObjetTableau.nombreObjets();`		
`ObjetTableau obj2 = new ObjetTableau(3);`		
`obj2.affiche();`		
`ObjetTableau.nombreObjets();`		

Eléments d'algorithmique

Maximum et minimum d'un tableau

Le tableau t est donné, ainsi que son nombre d'éléments n. On recherche soit le plus grand, soit le plus petit élément du tableau.

Algorithme de recherche du maximum
```
max ← t[0]
pour i allant de 1 à n-1
        si (t[i] > max) alors
            max ← t[i]
        fin si
fin pour
```

Algorithme de recherche du minimum
```
min ← t[0]
pour i allant de 1 à n-1
        si (t[i] < min) alors
            min ← t[i]
        fin si
fin pour
```

Indice du maximum et indice du minimum

Le tableau t est donné, ainsi que son nombre d'éléments n. On recherche la position du plus grand ou la position du plus petit élément du tableau.

Recherche de l'indice du maximum
```
max ← t[0]
indice ← 0
pour i allant de 1 à n-1
        si (t[i] > max) alors
            max ← t[i]
            indice ← i
        fin si
fin pour
```

Recherche de l'indice du minimum
```
min ← t[0]
indice ← 0
pour i allant de 1 à n-1
        si (t[i] < min) alors
            min ← t[i]
            indice ← i
        fin si
fin pour
```

Somme des éléments d'un tableau

Il s'agit de cumuler au fur et à mesure les valeurs des éléments du tableau. Il est donc nécessaire d'avoir une variable contenant les cumuls successifs.

Algorithme
```
som ← 0
pour i allant de 0 à n-1
    som ← som+t[i]
fin pour
```

A la fin, la somme des éléments du tableau est dans la variable som.

TD9 - Maximum, minimum, somme

1) Ouvrir le fichier **ObjetTableau.java** du dossier OBJETTABLEAU, compiler le fichier et le faire exécuter.

2) Ajouter à la classe Tableau deux méthodes d'instance `int maximum()` et `int minimum()` renvoyant respectivement le plus grand et le plus petit élément du tableau t.
Pour tester les méthodes, compléter le main de la classe ObjetTableau de la manière suivante :

```
Tableau obj1 = new Tableau(30) ;
obj1.affiche();
System.out.println("max : " + obj1.maximum() + " min : "+obj1.minimum()) ;
Tableau obj2 = new Tableau(50) ;
obj2.affiche();
System.out.println("max : " + obj2.maximum() + " min : "+obj2.minimum()) ;
```

3) Ajouter à la classe Tableau une méthode d'instance `int somme()` renvoyant **la somme** des éléments du tableau t.

Pour tester cette méthode, modifier la classe ObjetTableau en ajoutant au main les lignes suivantes :

```
System.out.println("Somme obj1 : " + obj1.somme()) ;
System.out.println("Somme obj2 : " + obj2.somme()) ;
```

Comptages divers

On a également besoin d'une variable supplémentaire lorsqu'on veut faire des comptages.
Exemple : compter le nombre d'éléments égaux à une valeur p

Algorithme

```
cpt ← 0
pour i allant de 0 à n-1
        si (t[i] vaut p) alors
                cpt ← cpt + 1
        fin si
fin pour
```

TD10 - Comptages, requêtes

1) Ajouter à la classe Tableau une méthode d'instance **`int egal(int p)`** comptant **le nombre d'éléments** de t qui sont **égaux à p**.
Pour tester cette méthode, ajouter au main les lignes :

```
System.out.println("obj1 : Nb d'elements egaux a 50 : " + obj1.egal(50));
System.out.println("obj1 : Nb d'elements egaux a 60 : " + obj1.egal(60));
System.out.println("obj2 : Nb d'elements egaux a 20 : " + obj2.egal(20));
System.out.println("obj2 : Nb d'elements egaux a 80 : " + obj2.egal(80));
```

2) Ajouter à la classe Tableau une méthode d'instance **`int compris(int p, int q)`** comptant **le nombre d'éléments** de t qui sont **strictement compris entre p et q (p < q)**.
A noter : l'égalité entre clauses logiques s'écrit &&.
Pour tester cette méthode, compter combien d'éléments de obj1 sont strictement compris entre 10 et 30 en ajoutant au main la ligne :

```
System.out.println("obj1: entre 10 et 30: " + obj1.compris(10,30));
```

3) Ajouter à la classe Tableau une méthode d'instance **`int premier(int p)`** renvoyant le **premier** élément de t **strictement supérieur à p**.
Pour tester cette méthode, déterminer le premier élément de obj1 strictement supérieur à 20 en ajoutant au main la ligne :

```
System.out.println("obj1 : premier element > 20: "+obj1.premier(20));
```

4) Ajoutez à la classe Tableau une méthode d'instance **`int dernier(int p)`** renvoyant le **dernier** élément de t **strictement inférieur à p**.
Pour tester cette méthode, déterminez le dernier élément strictement inférieur à 10 en ajoutant au main la ligne :

```
System.out.println("obj1 : dernier element < 10: "+obj1.dernier(10));
```

5) Ajouter à la classe `Tableau` la méthode d'instance **`int doublons()`** qui compte le nombre de fois où deux éléments **consécutifs** du tableau sont égaux.

Calcul de suites

Exemple déjà traité : le livret A (TD5).

TD11 : une suite récurrente

On considère la suite définie par : $U_0=0$ et $U_{i+1}=2U_i+3$. Créer un dossier nommé SUITE puis y écrire deux programmes :

1. un programme qui affiche les 20 premiers termes de cette suite ;
2. un programme qui affiche les termes de cette suite inférieurs à 100.

TD12 : la suite de Fibonacchi

Créer un dossier nommé FIBONACCHI, puis y écrire un fichier nommé Fibonacchi.java qui calcule et affiche les nombres (entiers) de la suite de Fibonacchi inférieurs à 100.
N.B. : Les deux premiers termes de la suite de Fibonacchi sont égaux à 1. Puis chaque terme est la somme des deux précédents.

Liste des algorithmes élémentaires :

- Somme
- Moyenne
- Recherche du minimum ou du maximum
- Recherche de l'indice du minimum ou du maximum
- Comptages
- Calcul de suites

TD13 : Quel algorithme utiliser ?

Connaissant le temps effectué par chaque cheval d'une course, afficher le numéro gagnant.
……..
Connaissant les bénéfices de toutes les filiales d'une entreprise, afficher celle qui est la plus rentable.
……..
Connaissant le chiffre d'affaire de chaque agence d'une entreprise, afficher le chiffre d'affaire total de l'entreprise.
……..
Afficher le nombre de caractères "e" contenus dans une phrase donnée.
……..
Connaissant les températures en Ile de France de chaque mois depuis 10 ans, afficher l'année dont la température moyenne est la plus froide.
……..
Connaissant la liste des coureurs du tour de France cycliste et les temps réalisés par chaque coureur à la 1e étape, afficher le temps du gagnant de cette étape.
……..

Permuter le contenu de deux variables

On veut permuter le contenu de deux variables a et b.
Exemple :

- Avant la permutation, a contient 38 et b contient 56.
- Après la permutation, a doit contenir 56 et b doit contenir 38.

Si on écrit a ← b (a reçoit la valeur de b), a prend la valeur de b **mais** la valeur précédente de a est perdue.

Il faut donc une variable temporaire où l'on stocke l'ancienne valeur de a.
On écrit donc :

temp ← a ; a ← b ; b ← temp ;

Pour permuter deux variables, on a donc **impérativement** besoin d'une **troisième**.

TD14 : inverser l'ordre des éléments d'un tableau

Ouvrir le fichier **ObjetTableau.java** du dossier OBJETTABLEAU.
Ajoutez à la classe `Tableau` la méthode d'instance `void inverse()` qui inverse l'ordre des éléments du tableau. Vous appliquerez votre méthode à l'objet `obj1`, puis vous afficherez `obj1` après modification en complétant la méthode main de la classe `ObjetTableau` avec les lignes de code suivantes :

```
obj1.inverse();
obj1.affiche();
```

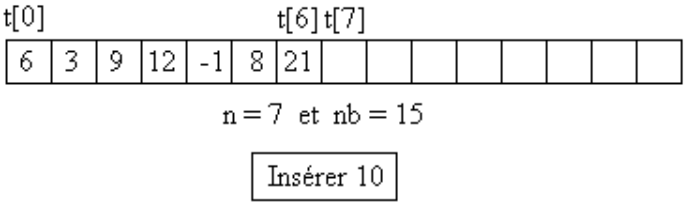
Par exemple, appliquée à un tableau contenant les valeurs ***9 ; 5 ; 7 ; 1 ; 4***, la méthode `inverse()` transformera le tableau avec les nouvelles valeurs suivantes ***4 ; 1 ; 7 ; 5 ; 9***.

Algorithmes d'insertion et de tri

Insérer un élément x dans un tableau à la suite des éléments existants

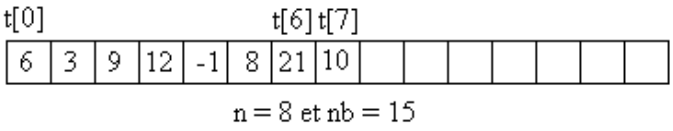
On suppose qu'un tableau t, de taille nb, contient n nombres, n ≤ nb.
Donc les cases t[0], t[1], …, t[n-1] du tableau sont donc remplies et les cases t[n], t[n+1], …, t[nb-1] sont vides.
On veut insérer un élément à la suite des éléments existants.

Exemple
Le tableau t dispose de 15 places (nb = 15) mais contient 7 nombres (donc n vaut 7), et on veut y insérer le nombre 10.



Algorithme :
si (n vaut nb)
alors
écrire « plus de place »
sinon
t[n] ← x
n ← n+1
fin si

Résultat :



TD15 : Insertion d'un élément à la fin d'un tableau

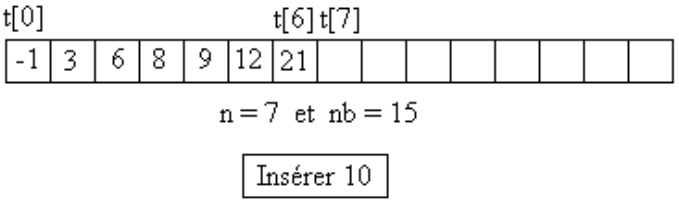
1) Se positionner dans le dossier PROGOBJET2. Compiler le fichier **ProgObjet2.java** puis le faire exécuter.

2) Ouvrir le fichier **ProgObjet2.java**. Regarder son code, puis compléter la méthode `void insere(int x)` afin d'insérer à la fin du tableau t l'entier x.

Insérer un élément à sa place, le tableau étant déjà rangé

Un tableau de nb places contient n nombres déjà rangés dans l'ordre croissant. On veut y insérer un nouveau nombre.

Exemple :



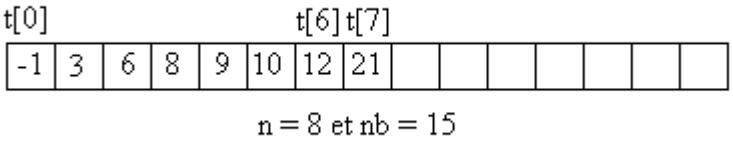
Principe
si le tableau est déjà plein, insertion impossible
si le tableau est vide, on met x dans t[0] et on met n à 1
sinon, c'est que le tableau est partiellement rempli :

- on cherche alors le premier indice i tel que t[i] > x
- on décale vers la fin du tableau les éléments d'indice allant de i à n-1 (**on commence par la fin**)
- on place x en t[i]
- on augmente n d'une unité

Algorithme

```
si (n vaut nb) alors
        écrire « Insertion impossible »
sinon
        si (n vaut 0) alors
                t[n] ← x
                n ← n + 1
        sinon
                i ← 0
                tant que ( (t[i] < x) et (i<n) )
                        i ← i + 1
                fin tant que
                pour j ← n-1 à i par pas de –1
                        t[j+1] ← t[j]
                fin pour
                t[i] ← x
                n ← n+1
        fin si
fin si
```

Résultat



n = 8 et nb = 15

TD16 : Insertion d'un élément dans un tableau rangé

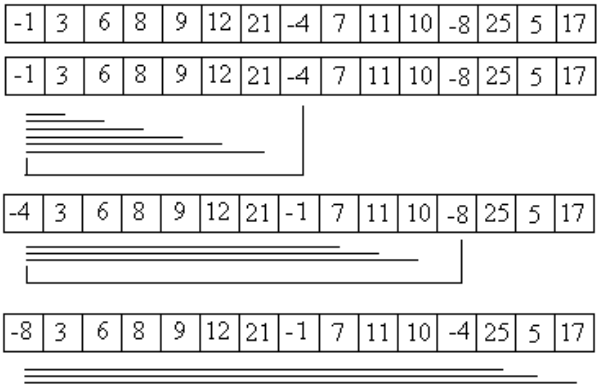
Ouvrir avec un éditeur le fichier **ObjetTableau2.java** qui se trouve dans le dossier OBJETTABLEAU2. Regarder son code, puis compléter la méthode `void insererange(int x)` afin d'insérer l'entier x à sa place dans le tableau.

Trier un tableau

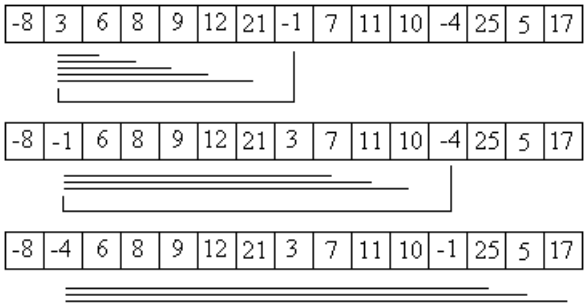
Tri tournoi

Principe
A partir d'un rang i, on considère l'élément t[i] du tableau et on le compare à tous les éléments t[j] suivants.
A chaque fois que t[i] est supérieur à t[j], on permute t[i] et t[j].
On effectue cela pour i allant du premier indice à l'avant-dernier indice.

Exemple
i = 0. On compare t[0] à t[1], t[2],…, t[n-1] et à chaque fois que t[0] > t[j], on permute t[0] et t[j].
A la fin de ce premier parcours, le plus petit élément est arrivé en première position :



On recommence avec i = 1 : on compare t[1] à t[2], … , t[n-1] et lorsque t[1] > t[j], on permute t[1] et t[j].
A la fin du deuxième parcours, les deux plus petits éléments sont aux deux premières places :



On recommence ensuite avec i = 2, 3, … n-2.

```
pour i ← 0 à n-2
        pour j ← i+1 à n-1
                si (t[i] > t[j]) alors
                        temp ← t[i]
                        t[i] ←  t[j]
                        t[j] ←  temp
                fin si
        fin pour
fin pour
```

Tri à bulles

Principe

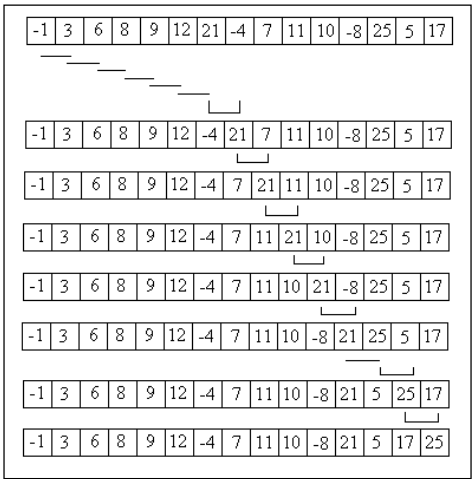
On compare chaque élément à son suivant.

S'il lui est supérieur, on les permute ; sinon, on passe à l'élément suivant.

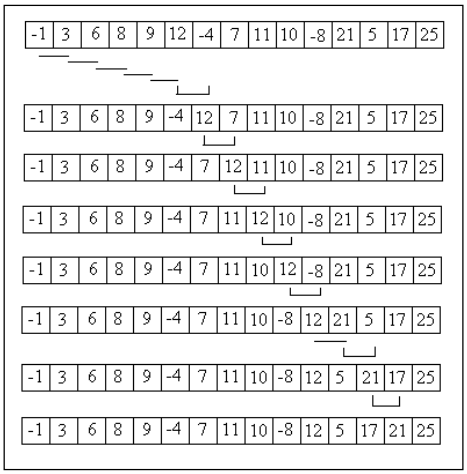
Lorsqu'on a parcouru tout le tableau, on recommence un nouveau parcours s'il y a eu au moins une permutation.

Exemple du premier parcours : ci-contre

A la fin de ce premier parcours, le plus grand élément est arrivé en dernière position.



Au parcours suivant, on obtient le résultat ci-contre :



Les deux plus grands éléments sont en avant-dernière position et dernière position.

Le tableau est trié dès qu'un parcours est fait sans aucune permutation.

Pour contrôler l'arrêt des permutations, on utilise une variable initialisée à 0 au début de chaque parcours, et que l'on passe à 1 dès que l'on fait une permutation.

Il suffit alors de boucler tant que cette variable, à la fin d'un parcours, est égale à 1.

Algorithme
permut ← 1
tant que (permut vaut 1)
permut ← 0
pour i ← 0 à n-2
si (t[i] > t[i+1]) alors
permut ← 1
temp ← t[i]
t[i] ← t[i+1]
t[i+1] ← temp
fin si
fin pour
fin tant que

TD17 : Tri d'un tableau

Ouvrir le fichier **ObjetTableau3.java** qui se trouve dans le dossier OBJETTABLEAU3. Ajouter les deux méthodes `void tri1()` et `void tri2()` implémentant le tri tournoi et le tri à bulles. Les tester séparément en insérant d'abord dans le main les instructions `obj1.tri1();obj1.affiche();` puis les instructions `obj1.tri2();obj1.affiche();`

Les paquetages Java

Qu'est-ce qu'un paquetage ?

Les paquetages (packages) sont des librairies de **classes,** de classes d'**exceptions** et d'**interfaces** regroupées selon leur fonction, qui sont fournies en même temps que le compilateur javac et l'interpréteur java.
La documentation concernant chaque paquetage est accessible : **http://docs.oracle.com/javase/8/docs/api**

Les classes et la notion d'héritage

Les classes des différents paquetages sont liées par la notion **d'héritage**.
On déclare qu'une classe dérive d'une autre classe par :

```
class classe-fille extends classe-mère {
...
}
```

Si une classe B dérive d'une classe A, elle hérite des champs et des méthodes de A.
Un objet de la classe fille est donc un objet de la classe mère (éventuellement complété par des champs et des méthodes supplémentaires), mais la réciproque est fausse.
Toutes les classes héritent de la classe **Object**. Si aucune clause d'héritage n'est spécifiée dans la définition d'une classe, par défaut, elle hérite de la classe Object.

TD18

1. Etude du fichier UtilPoint.java

On considère l'application suivante enregistrée dans le fichier UtilPoint.java.
Elle comporte la classe PointColore qui dérive de la classe Point.

```
class Point{
   int x,y,z;
   Point(int a, int b, int c){
      x=a;
      y=b;
      z=c;
      }
   void affiche(){
         System.out.println(x+" "+y+" "+z);
      }
   }

class PointColore extends Point{
   int color;       //attribut supplémentaire
   PointColore(int a, int b, int c, int d){    //le constructeur à 4 paramètres
      super(a,b,c);                            //fait appel à celui de Point (super fait référence à la classe mère)
      color=d;                                 //et ajoute l'initialisation de la couleur
      }
   void affiche(){                             //on redéfinit affiche
      super.affiche();                         //en faisant appel à affiche() de la classe mère
      System.out.println(color);               //et en complétant par l'affichage de la couleur
      }
  }

public class UtilPoint{
   public static void main(String args[]){
      Point p=new Point(1,2,3);
      p.affiche();
      PointColore q=new PointColore(4,5,6,25);
      q.affiche();
      }
   }
```

2. Ecriture du fichier Util3D.java

Créer le dossier Util3D puis écrire le fichier Util3D.java qui comporte deux classes :
- Point2D qui définit des objets à deux coordonnées et comporte une méthode affiche()
- Point3D qui dérive de Point2D et ajoute une 3e coordonnée. Ecrire pour la classe Point3D une méthode affiche() qui fait appel à celle de la classe mère et affiche les trois coordonnées.

Protection des attributs et méthodes

Les attributs et méthodes peuvent être déclarés **private**, **public**, **protected** ou ne pas être précédés de déclaration de protection.

Les attributs et méthodes déclarés **private** ne sont accessibles qu'à l'intérieur de leur propre classe.
→ a ne sera donc accessible qu'aux méthodes de la classe Truc.

Les attributs et méthodes déclarés **protected** ne sont accessibles qu'aux sous-classes (qu'elles soient dans le même paquetage ou pas) et aux classes du même paquetage.
→ b sera donc accessible aux sous-classes de Truc et aux classes du même paquetage que Truc.

Les attributs et méthodes déclarés **public** sont toujours accessibles.
→ c sera donc toujours accessible

Les attributs et méthodes **sans aucune déclaration de protection** sont accessibles par toutes les classes du même paquetage. C'est le mode de protection par défaut.

→ d sera donc accessible aux classes du même paquetage que Truc.

TD19 - Protection des attributs

Ouvrir le fichier **UtilTruc.java** du dossier **UtilTruc**.
Son contenu est le suivant :

```
class Truc {
     private int a ;
     protected int b ;
     public int c ;
     int d ;

     public Truc(int u, int v, int w, int x) {
           a = u ;
           b = v ;
           c = w ;
           d = x ;
           }

     public void affiche() {
           System.out.println("----------------------------------------");
           System.out.println("affiche de la classe Truc : ");
           System.out.println("a (private) : " + a);
           System.out.println("b (protected) : " + b);
           System.out.println("c (public) : " + c);
           System.out.println("d (friendly) : " + d);
           System.out.println("FIN de affiche de la classe Truc");
           System.out.println("----------------------------------------");
           }
     }
public class UtilTruc {
     public static void main(String args[]) {
           System.out.println("main de la classe UtilTruc : ");
           System.out.println("on cree l'objet t de coordonnees
                 1,2,3,4");
           Truc t = new Truc(1,2,3,4);
           t.affiche();

           System.out.println("main de la classe UtilTruc (suite):");
           System.out.println("valeur de a (private) : " + t.a);
           System.out.println("valeur de b (protected) : " + t.b);
           System.out.println("valeur de c (public) : " + t.c);
           System.out.println("valeur de d (friendly) : " + t.d);

           System.out.println("dans le main, on veut CHANGER t");
           t.a = 5 ;
           t.b = 6 ;
           t.c = 7 ;
           t.d = 8 ;
           t.affiche();

           System.out.println("FIN du main de la classe UtilTruc");
           }
     }
```

Compilez ce fichier. Quelles sont les instructions interdites ? Pourquoi ? Notez les explications :

……
……

```
……
……
```

Barrez-les dans le programme ci-dessus et commentez-les dans le fichier.
Lorsque le programme se compile, complétez ci-dessous le résultat de l'exécution :

```
main de la classe UtilTruc :
on cree l'objet t de coordonnees 1, 2, 3, 4
----------------------------------------
affiche de la classe Truc :
a (private) :
b (protected) :
c (public) :
d (friendly) :
FIN de affiche de la classe Truc
----------------------------------------
main de la classe UtilTruc (suite) :
valeur de b (protected) :
valeur de c (public) :
valeur de d (friendly) :
dans le main, on veut CHANGER t
----------------------------------------
affiche de la classe Truc :
a (private) :
b (protected) :
c (public) :
d (friendly) :
FIN de affiche de la classe Truc
----------------------------------------
FIN du main de la classe UtilTruc
```

Les interfaces

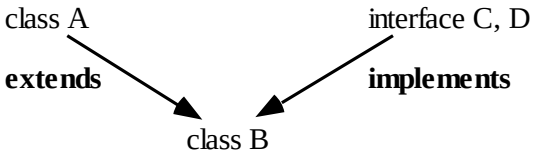
Une interface est une sorte de classe spéciale, dont toutes les méthodes sont déclarées public et abstract (sans code). En fait, une interface **propose des services**, mais sans donner le code permettant d'exécuter ces services.
Une classe qui souhaite utiliser une interface le signale par le mot **implements** :

```
class B implements C { ... } ;
```

On dit alors qu'elle **implémente** l'interface. Dans la classe B, on doit alors obligatoirement donner le code de **toutes** les méthodes de l'interface.

A noter : une classe B peut hériter d'une classe A et implémenter une ou plusieurs interfaces :

```
class B extends A implements C, D {...} ;
```



Les méthodes des interfaces C et D doivent alors **obligatoirement** être définies dans la classe B.

Les classes d'exceptions

Lorsqu'une méthode est exécutée, il peut se produire des erreurs. On dit alors qu'une **exception** est **levée**. Une exception est un objet, instance d'une des nombreuses sous-classes de la classe **Exception**.
Chaque méthode qui risque de provoquer une ou des erreur(s) le signale par le mot **throws** suivi de la nature des erreurs susceptibles de se produire.
Lorsqu'on veut utiliser une méthode signalée par **throws**, on englobe l'appel à cette méthode dans un bloc **try** (pour capturer l'erreur) et on précise ce qu'on fera en cas d'erreur dans une clause **catch.**

Exemple :

```
try   {
      fichier.read() ;
      }
catch (IOException) {
      System.out.println(" erreur de lecture ") ;
      return ;
      }
```

Les différentes classes d'erreurs possibles sont spécifiées dans chaque paquetage.

java.lang

Le paquetage java.lang est le **seul** paquetage dont l'emploi ne doit **jamais** être déclaré.
Il comprend notamment :

- la classe **Object**
- la classe **System**
- les **classes enveloppantes**
- la classe **Math**
- la classe **String** (classe des chaînes de caractères).

La classe **Object** est à l'origine de toutes les classes. Elle contient une méthode d'instance appelée **`toString()`** dont hérite donc tout objet. Cette méthode permet de convertir (au mieux) l'objet en une chaîne de caractères affichable.

La classe **System** contient des variables et des méthodes du système. Quand on écrit `System.out.println(...)`, on fait appel à la méthode `println` de la variable de classe **`out`** de la classe System.

Les classes enveloppantes : les variables des types de données primitifs (int, char, double, etc) sont les seuls éléments qui ne sont **pas** des objets dans Java. Mais on peut, en cas de besoin, les transformer en objets à l'aide de classes enveloppantes suivantes :

type	nature	classe enveloppante
boolean	true ou false	Boolean
char	caractère 16 bits	Character
byte	entier 8 bits signé (petit entier)	Byte
short	entier 16 bits signé (entier court)	Short
int	entier 32 bits signé (entier)	Integer
long	entier 64 bits signé (entier long)	Long
float	virgule flottante 32 bits (réel simple)	Float
double	virgule flottante 64 bits (réel double)	Double

Exemples d'utilisation :

```
int n = 5 ;                              // n n'est pas un objet
Integer obj1 = new Integer(n) ;          // obj1 est un objet

double f = 7.2563 ;                      // f n'est pas un objet
Double obj2 = new Double(f) ;            // obj2 est un objet
```

La classe **Math** contient les **méthodes de classe** nécessaires pour effectuer les calculs mathématiques.
Exemples :

```
double j = -3.145 ;
double k = Math.abs(j) ;                 // valeur absolue
double l = Math.pow(j,k) ;               // j exposant k
double m = Math.sqrt(l) ;                // racine carrée de l
```

La classe **String** contient de très nombreuses **méthodes d'instance** pour le traitement des chaînes de caractères.

Conversions :

- **Nombre vers chaîne de caractères :**

  1. **entier vers chaîne de caractères**
     ```
     int i = 23 ;
     String si ;
     si = new Integer(i).toString();
     ```

  2. **float vers chaîne de caractères**
     ```
     float f = 23.1 ;
     String sf ;
     sf = new Float(f).toString();
     ```

  3. **double vers chaîne de caractères**
     ```
     double d = 34.2 ;
     String sd ;
     sd = ……
     ```

- **Chaîne de caractères vers nombre :**

  1. **chaîne de caractères vers entier**
     ```
     String si = "23";
     int i;
     i = new Integer(si).intValue();
     ```

  2. **chaîne de caractères vers float**
     ```
     String sf = "23.1";
     float f;
     f = new Float(sf).floatValue();
     ```

  3. **chaîne de caractères vers double**
     ```
     String sd = "45.123";
     double d;
     d = ……
     ```

TD20 - La classe String

Le dossier CHAINES contient le fichier **Chaine.java** :

```
public class Chaine {
     public static void main(String args[]) {
           String s1 = "Bonjour" ;
           System.out.println("s1 vaut "+s1+" et est de longueur  " +
           s1.length());

           char c = s1.charAt(3) ;
           System.out.println("Le caractere au 3eme indice est : " +c) ;

           String s2 = s1.toUpperCase();
           if (s1.equalsIgnoreCase(s2)) {
                 System.out.println("s1 et s2 sont equivalents");
                 }

           String s3=" et bonne journee" ;
           String s4 = s2.concat(s3) ;
           System.out.println("s4 vaut : " + s4);
           String s5 = "bon" ;
           int pos = s4.indexOf(s5) ;
           if (pos == -1) {
                 System.out.println("s4 ne contient pas s5");
                 }
```

```
            else {
                System.out.println("s4 contient s5 a l'indice  "+pos);
                }

           String s6 = s4.substring(3, 8);
           System.out.println("entre les indices 3 et 8 de s4 on trouve "
           + s6);
           }
     }
```

1) Faire exécuter ce fichier et noter les résultats ci-dessous.

```
s1 vaut               et est de longueur
Le caractere au 3eme indice est :
s1 et s2 sont ...
s4 vaut :
s4 contient s5 a l'indice ...
entre les indices 3 et 8 de s4 on trouve ...
```

2) Rechercher dans la documentation ce que fait chacune des méthodes d'instances suivantes :

`length() :`

`charAt(int n) :`

`toUpperCase() :`

`equalsIgnoreCase(String s) :`

`concat(String s) :`

`indexOf(String s) :`

`substring(int n, int m) :`

java.text

Le paquetage java.text contient des classes utilitaires.
Il contient notamment les classes permettant le formatage de textes à afficher, dont :

- les classes **Format, NumberFormat, DecimalFormat** pour le formatage de nombres
- la classe **DateFormat** pour le formatage des dates.

A noter : Pour utiliser tout ou partie d'un paquetage quelconque **autre que java.lang**, on doit déclarer l'usage de ce paquetage par le mot-clé `import`

Exemples pour le paquetage java.text :

`import java.text.Format ;` → on se donne le droit d'utiliser la classe Format
`import java.text.NumberFormat ;` → on se donne le droit d'utiliser la classe NumberFormat
`import java.text.* ;` → on se donne le droit d'utiliser toutes les classes du paquetage

TD21 - Classes Format pour les nombres décimaux

Ouvrez le fichier **AffNombre.java** du dossier AFFNOMBRE. Il a le contenu suivant :

```
import java.text.* ;  // noter l'importation du paquetage

public class AffNombre {
     public static void main(String args[]) {
           double d = 56125.458997 ;
           System.out.println("--------------------------");
```

```
            System.out.println("Affichage de d sans traitement : " + d);
            System.out.println("--------------------------");

            DecimalFormat df = new DecimalFormat() ;
            DecimalFormatSymbols dfs = df.getDecimalFormatSymbols();
            dfs.setGroupingSeparator('.') ;
            df.setDecimalFormatSymbols(dfs);
            String s0 = df.format(d) ;
            System.out.println("utilisation format avec pt decimal:"+s0) ;
            System.out.println("--------------------------");

            dfs.setGroupingSeparator(' ') ;
            df.setDecimalFormatSymbols(dfs);
            String s1 = df.format(d) ;
            System.out.println("utilisation format avec espace : "+s1) ;
            System.out.println("--------------------------");

            df.applyPattern("#,###,##0.00") ;
            String s2 = df.format(d) ;
            System.out.println("utilisation format et pattern : "+s2) ;
            System.out.println("--------------------------");

            System.out.println("");
            double f = 0.53 ;
            System.out.println("Affichage de f sans traitement : " + f);
            System.out.println("--------------------------");

            df.applyPattern("#,###,##0.00 euros") ;
            String s3 = df.format(f);
            System.out.println("pattern avec unite monetaire : " + s3);
            System.out.println("--------------------------");

            df.applyPattern("#,###,##0.00 %") ;
            String s4 = df.format(f);
            System.out.println("pattern avec format pourcentage : "+s4) ;
            System.out.println("--------------------------");
            }
    }
```

1) Compilez-le, exécutez-le et notez à côté des lignes correspondantes les affichages obtenus

```
Affichage de d sans traitement :
utilisation format avec pt decimal :
utilisation format avec espace :
utilisation format et pattern :
Affichage de f sans traitement :
pattern avec unite monetaire :
pattern avec format pourcentage :
```

2) Ajoutez à la fin de la méthode main les lignes suivantes :

```
df.applyPattern(" ... ") ;          // ligne 1
String s5 = df.format(d) ;
System.out.println("exercice pattern : "+s5) ;
System.out.println("--------------------------");
```

et complétez le pattern de la ligne 1 afin d'obtenir : `exercice pattern : 56125,46`

java.util

Le paquetage java.util contient des classes utilitaires, notamment :

- **Calendar** et **GregorianCalendar** (fille de Calendar) : pour les dates
- **Random** : pour les nombres aléatoires
- **Vector** et **Enumeration** : pour le stockage et la récupération d'objets en nombre illimité.

TD22 - Classe GregorianCalendar et calcul de l'âge

1) Faire exécuter le fichier **Anniversaire.java** du dossier ANNIVERSAIRE.
La date du jour est déterminée par un appel à un constructeur de la classe GregorianCalendar : `new GregorianCalendar()`.

```
import java.util.* ;

public class Anniversaire {
     public static void main (String args[]) {
           GregorianCalendar gc = new GregorianCalendar() ;

           int jour = gc.get(Calendar.DATE) ;
           int mois = gc.get(Calendar.MONTH)+1 ;
           int annee = gc.get(Calendar.YEAR) ;
           System.out.println("Nous sommes le "+jour+"/"+ mois+"/"+
                 annee);

           // Votre date de naissance
           int j = 23 ;                  // ligne 1
           int m = 8 ;                   // ligne 2
           int a = 1988 ;                // ligne 3
           System.out.println("Vous etes ne(e) le "+j+"/"+m +"/"+a) ;

           // ici : calculez votre age (annees seulement)
           ...
           ...
           ...
           }
     }
```

2) Remplacez la date donnée par les lignes 1 à 3 par votre date de naissance, puis complétez le fichier afin de faire afficher « Cette annee, vous avez … ans. ».

TD23 - Classe Random et tirage d'un loto

1) Faire exécuter le fichier **CreeLoto.java** du dossier LOTO.
Un objet de la classe Loto contient un tableau de 6 entiers tirés aléatoirement (entre 1 et 49 compris) par les instructions :

```
int m = 1 + r.nextInt(49);
t[i] = m;
```

Par contre, l'unicité de chaque nombre n'est pas (encore) vérifiée.

```
import java.util.Random ;

class Loto {          // début de la classe
     int[] t ;
     Random r ;

     Loto() {    // le constructeur
           t = new int[6] ;  // on réserve la place pour le tableau t
           r = new Random() ;
```

```
            int i = 0 ;
            while (i < 6) {
                  int m = 1 + r.nextInt(49);          // ligne 1


                  t[i] = m;                           // ligne 2
                  i++ ;
                  }
            }                 // fin du constructeur

      void affiche() {
            for (int i = 0; i < 6; i++){
                  System.out.print(t[i]+" * ");
                  }
            System.out.println("\n");
            }
      }

public class CreeLoto {
      public static void main (String args[ ]) {
            Loto l = new Loto() ;
            l.affiche();
            }
      }
```

2) Modifier ce fichier afin qu'**aucun nombre ne soit répété** lors du tirage. Pour cela, créer et introduire le test adéquat entre ligne 1 et ligne 2 :

```
            while (i < 6) {
                  int m = 1 + r.nextInt(49);          // ligne 1
                  ...
                  ...
                  t[i] = m;                           // ligne 2
                  i++ ;
                  }
```

TD24 - Classes Vector et Enumeration

1) Un objet de la classe **Vector** peut stocker des **objets** de tous types **en nombre illimité**. Regardez la documentation concernant cette classe. **A noter** : On ajoute un objet à un Vector par la méthode add(Object).

2) Ouvrez le fichier **CreeVecteur.java** du dossier VECTEUR. Il utilise le mécanisme d'héritage pour créer une classe Vecteur, fille de la classe Vector.

A noter :
Le constructeur de Vecteur fait appel au constructeur de Vector par l'instruction `super()`.
Tout objet de la classe Vecteur **hérite** ainsi des attributs et méthodes de la classe Vector.
Mais dans cette classe Vecteur, de nouvelles méthodes (`remplit` et `affiche1`) sont ajoutées.

```
import java.util.* ;

class Vecteur extends Vector {     // début de la classe, fille de Vector
      Vecteur() {                  // le constructeur
            super() ;
            }                      // fin du constructeur

      void remplit() {
            Random r = new Random();
            int n = r.nextInt(20) + 1 ;

            for (int i = 0 ; i < n ; i++)
```

```
                this.add(new Integer(r.nextInt(100)).toString());
          }

     void affiche1() {
          for (Enumeration e = this.elements(); e.hasMoreElements();)
                System.out.print(e.nextElement() + " * ");
          System.out.println();
          }
     }

public class CreeVecteur {
     public static void main (String args[ ]) {
          Vecteur v = new Vecteur() ;
          v.remplit();
          v.affiche1() ;
          }
     }
```

Compilez le fichier, faites-le exécuter et exprimez en une phrase ce que fait cette application :
……
……
……

3) Etudiez de près les deux méthodes `remplit` et `affiche1`. Dans la méthode `affiche1`, on utilise un objet de la classe **Enumeration** pour parcourir le Vector. Notez donc la syntaxe très particulière de la boucle `for` de `affiche1`.

4) En utilisant la documentation relative à la classe **Vector**, indiquez la signification de :
- la variable d'instance **elementCount** :

……
- la méthode **elementAt(int index)** :

……

Utilisez **elementCount** et **elementAt** pour écrire une méthode `affiche2` équivalente à la méthode `affiche1`. Puis testez la méthode `affiche2` dans le `main`.
5) Ajoutez à la classe Vecteur une méthode `afficheCarre1()` qui affiche les carrés des entiers contenus dans un vecteur, séparés par des étoiles. Utilisez une boucle `for` identique à celle de la méthode `affiche1`.
Pour tester votre méthode, ajoutez au main la ligne : `v.afficheCarre1() ;` compilez et exécutez.

6) Ajoutez à la classe Vecteur une méthode `afficheCarre2()` qui doit être équivalente à la méthode `afficheCarre1()` et utiliser une boucle `for` identique à celle de la méthode `affiche2`.
Testez la méthode `afficheCarre2` dans le `main`.

java.io

Le paquetage **java.io** contient :
- des **classes utilitaires** pour la lecture à partir du clavier
- des **classes utilitaires** pour la lecture et l'écriture de fichiers
- diverses **interfaces**, dont l'interface Serializable qui permet l'enregistrement d'objets dans un fichier.

TD25 - Lecture au clavier

1) Compilez et testez le fichier **UtilChaine.java** du dossier UTILCHAINE. Il permet la lecture d'une chaîne de caractères au clavier, puis son affichage. Il fait appel aux classes `BufferedReader` et `InputStreamReader` de java.io.

```
import java.io.* ;
class Chaine {
```

```
     String ch1 ;
     String ch2 ;
     Chaine() {
          try {                    // lecture de ch1 au clavier
           BufferedReader in = new
                 BufferedReader(new InputStreamReader(System.in));
           ch1 = in.readLine() ;
           ch2 = "" ;                    // provisoirement
           }
          catch (IOException e) {
                 System.out.println("erreur d'entrée/sortie") ;
                 ch1 = "" ;              // chaîne vide
                 return ;
                 }
           ...                           // initialisation de ch2
           ...
           }

     void affiche() {
           System.out.println(ch1) ;
           System.out.println(ch2) ;
           }
     }

public class UtilChaine {
     public static void main(String args[]) {
           Chaine s = new Chaine() ;
           s.affiche() ;
           }
     }
```

2) Complétez-le afin que ch2 soit égal à la chaîne ch1 à l'envers. On consultera la documentation de la classe String (dans java.lang) et on utilisera un tableau auxiliaire de caractères, de type char tab[ ].

3) Ajoutez à la classe Chaine une méthode `boolean palindrome()` qui teste si ch1 est un palindrome (si ch1 et ch2 ont le même contenu) et renvoie soit true soit false.

Pour tester la méthode, complétez le main par les lignes :

```
if (s.palindrome())
      System.out.println("palindrome") ;
      else System.out.println("pas palindrome") ;
```

Enregistrez à nouveau le fichier UtilChaine.java, compilez-le et faites-le exécuter.

TD26 - Lecture d'un fichier

1) Ouvrez le fichier **Lecture.java** du dossier LECTURE. La classe principale Lecture demande la création d'un objet de la classe Lecteur. Tout objet de la classe Lecteur contient un entier appelé nb et un tableau de chaînes de caractères appelé contenu pouvant contenir jusqu'à 100 chaînes de caractères.

La méthode `void lit(String nom)` permet de lire un fichier de texte et de remplir le tableau contenu. Elle fait appel à des classes du paquetage java.io (`BufferedReader, FileReader`).

La méthode `readLine()` utilisée de la classe `BufferedReader` est déclarée avec la clause **throws** (elle risque de provoquer des erreurs en cas d'absence du fichier ou de fichier endommagé). Elle ne peut donc être utilisée que dans un bloc `try {...} catch (Exception e) {...}`.

2) Compilez et exécuter le fichier.

```
import java.io.* ;
class Lecteur {
     String[] contenu ;
     int nb ;
     Lecteur() {
           contenu = new String[100] ;
           }
     void lit(String nom) {
           String ligne ;
           BufferedReader lecteurAvecBuffer ;
           nb = 0 ;
           try   {
                 lecteurAvecBuffer =
                       new BufferedReader (new FileReader(nom));
                 while ((ligne = lecteurAvecBuffer.readLine()) != null) {
                       contenu[nb] = ligne;
                       nb++ ;
                       }
                 lecteurAvecBuffer.close();
                 }
           catch (Exception e) {
                 System.out.println("Erreur de fichier");
                  }
           }
     void affiche() {
           for (int i = 0 ; i < nb ; i++) {
                 System.out.println(contenu[i]);
                 }
           }
     }

public class Lecture {
     public static void main(String args[]) {
           Lecteur l = new Lecteur();
           l.lit("verlaine.txt") ;
           System.out.println("Poeme de Verlaine");
           l.affiche() ;                                    // ligne 1
           System.out.println("----------------------");
           System.out.println("Poeme de Hugo");
           l.lit("hugo.txt") ;
           l.affiche() ;                                    // ligne 2
           }
     }
```

3) Ajoutez à la classe Lecteur une méthode `int compteLettre(char c)` qui détermine le nombre de fois que le caractère c est utilisé dans le texte lu.
Pour la tester, compléter le main de la classe Lecture par la ligne :
`System.out.println("Nombre de a : " + l.compteLettre('a')) ;`
insérée juste après la ligne 1 ET juste après la ligne 2, afin de compter le nombre de 'a' dans chacun des deux poèmes.

TD27 - Sérialisation

La sérialisation permet de stocker dans un fichier l'état des objets créés dans un programme et de les recréer plus tard. Un objet peut ainsi exister entre deux exécutions d'un programme, ou entre deux programmes : c'est la **persistance objet**.

Pour rendre un objet persistant, il faut que sa classe implémente l'interface **java.io.Serializable**.

Dans le programme **TestSerializable.java** du dossier SERIALIZABLE, les objets sérialisés sont enregistrés sur le disque dans un fichier nommé « fichier ». C'est l'appel dans le main à la méthode `str.writeObject()` qui lance la sérialisation.

```
import java.io.* ;

class Objet implements Serializable {
     int a ;
     int b ;
     int c ;

     Objet(int p, int q, int r) {        // le constructeur
           a = p ;
           b = q ;
           c = r ;
           }                             // fin du constructeur
     void affiche() {
           System.out.println(a + " " + b + " " + c);
           }
     }

public class TestSerializable {
     public static void main (String args[ ]) {
           try {
                 Objet objet1 = new Objet(4, 8, 12) ;
                 Objet objet2 = new Objet(-3, 2, 7) ;
                 Objet objet3 = new Objet(11, -5, 8) ;

                 FileOutputStream f = new FileOutputStream("fichier");
                 ObjectOutputStream str = new ObjectOutputStream(f);
                 str.writeObject(objet1);
                 str.writeObject(objet2);
                 str.writeObject(objet3);

                 str.close();
                 }
           catch(Exception e) {
                 System.out.println("fichier non cree");
                 }
           }
     }
```

1) Compiler et faire exécuter ce fichier. Vérifier qu'un fichier appelé « fichier » est effectivement créé dans le dossier.

2) Le programme **DeSerializable.java** du même dossier permet de récupérer les objets sérialisés dans ce fichier nommé « fichier ». C'est l'appel dans le main à la méthode `str.readObject()` qui lance la désérialisation.

```
import java.io.* ;

class Objet implements Serializable {   // début de la classe
     int a ;
     int b ;
     int c ;

     Objet(int p, int q, int r) {  // le constructeur
           a = p ;
           b = q ;
           c = r ;
           }                       // fin du constructeur

     void affiche() {
           System.out.println(a + " " + b + " " + c);
           }
     }
```

```
public class DeSerializable {
     public static void main (String args[]) {

           try {
                 FileInputStream f = new FileInputStream("fichier");
                 ObjectInputStream str = new ObjectInputStream(f);

                 Objet objet1 = (Objet)str.readObject();
                 Objet objet2 = (Objet)str.readObject();
                 Objet objet3 = (Objet)str.readObject();
                 objet1.affiche();
                 objet2.affiche();
                 objet3.affiche();

                 str.close();
                 }
           catch(Exception e) {
                 System.out.println("fichier non trouve");
                 }
           }
     }
```

3) Compiler et exécuter ce fichier. Vérifier que vous retrouvez bien les objets précédemment sérialisés.

4) Modifier le fichier **CreeLoto.java** du dossier LOTO pour qu'un objet de la classe Loto soit sérialisable. Dans le `main`, créez cinq lotos différents et sérialisez-les.

5) Ecrivez ensuite un autre programme affectuant la désérialisation et permettant d'afficher les uns en-dessous des autres ces cinq lotos.

TD28 : utilisation de java.io et java.util

Ouvrir le fichier **Devinette.java** qui se trouve dans le dossier DEVINETTE.
Compléter cette application pour qu'elle demande à l'utilisateur de deviner un nombre entier généré aléatoirement entre 1 et 100. Le programme indique à l'utilisateur si son nombre est trop haut ou trop bas et ne s'arrête que s'il a deviné juste. Le programme donne finalement le nombre de coups tentés pour obtenir le succès.

TD29 : utilisation de java.io et de la classe String

Ouvrir le fichier **MessageINSEE.java** qui se trouve dans le dossier INSEE.

1. Quel est le rôle de la méthode `isLong` ?

…….

2. Compléter cette application pour qu'elle demande à l'utilisateur (né avant l'année 2000) son numéro INSEE et affiche par exemple le message suivant :

Bonjour **Madame**. D'après votre numéro INSEE, vous êtes né**e** dans le département **93** au mois de **juillet 1997**.
Les parties en gras sont les parties à déduire du numéro INSEE.
On pensera à tester que le texte entré est bien un nombre, qu'il comporte le bon nombre de caractères, et que les valeurs sont cohérentes (il n'y a par exemple que 12 mois dans l'année).
On utilisera des tableaux de chaînes de caractères.

Annexe

Gestion des caractères accentués dans les fichiers java
Au besoin compiler avec :
javac –encoding UTF-8 *fichier.java*
L'éditeur NotePad++ utilise le codage UTF-8 par défaut.