# TP1 : Données structurées / Manipulation de fichiers de données

Le TP sera réalisé en Java.

## 1. Introduction

On considère une série d'enregistrements concernant des ventes réalisées par un exportateur de véhicules miniatures. Pour chaque vente, il entre dans son registre de nombreuses informations :

- nom de la société cliente
- nom et prénom du contact, adresse, téléphone
- nombre d'unités vendues
- prix de vente
- etc...

Ces informations sont stockées dans un fichier au format 'csv' (comma separated values) : `ventes.csv` Récupérez ce fichier sur Claroline.

Dans un premier temps, regardez son contenu avec un editeur de texte (**geany**, **gedit** ou autre...). La première ligne contient les noms des attributs (`NUM_COMMANDE`, `QUANTITE`,...). Les ligne suivantes contiennent les valeurs d'attributs correspondant à une vente donnée. en tout plus de 2000 ventes sont répertoriées dans ce fichier.

Ouvrez-le maintenant à l'aide d'un tableur (par exemple **localc**). Les données sont maintenant "rangées" en lignes et colonnes pour faciliter la lecture.

Notez bien l'emplacement du fichier `ventes.csv` dans l'arborescence de fichiers.

## 2. Lecture de fichier texte et traitement des données

Lancez le programme **Eclipse** et définissez un nouveau projet.

**Rappel : utilisation du logiciel Eclipse**

**Etape 1** : créer le répertoire de travail et le projet Au lancement d'Eclipse, choisissez un répertoire de travail (vous pouvez fermer la fenêtre d'accueil). Dans l'onglet « File », sélectionnez « New », puis « Project ». Choisir l'option « Java Project », puis cliquez sur « Next ». Donnez un nom à votre projet (Par exemple « TP1 »), ensuite appuyez sur « Next », puis sur « Finish ». Votre projet est maintenant créé.

**Etape 2** : créer le/les fichier(s) du projet Un projet java est constitué de plusieurs classes permettant de définir des "objets". A chaque fois que vous devez créer une nouvelle classe, allez dans l'onglet « File », puis « New » puis « Class ». Donnez à votre fichier le nom de votre classe. Par exemple, "Principal", "Client", "Departement",... S'il s'agit de la classe principale, cochez la case « public static void main (String[] args) » afin que la classe contienne une méthode "main" qui sert de point de démarrage de votre programme. Cliquez sur « Finish ». Ca y est, le fichier est créé avec une classe prête à être complétée.

**Etape 3** : Une fois le programme écrit, vous pourrez exécuter l'application en choisissant l'onglet « Run », puis « Run As », puis « Java Application ».

### 2.1 Ouverture du fichier

La lecture dans un fichier s'effectue avec la librairie `java.io` où sont définies les classes `FileReader` et `BufferedReader` :

```
import java.io.*;
```

Pour pouvoir lire un fichier, il faut connaître son chemin d'accès, noté ici `"/chemin/vers/le/fichier"`. En Java, ce chemin est nécessaire pour créer un flux de lecture de type `FileReader`. Pour accéder au contenu du fichier, il faut définir un objet de type `BufferedReader` servant à charger dans un "tampon" les données du fichier (le "buffer").

```
FileReader f = null;
BufferedReader g = null;
```

Il est important de vérifier que l'opération d'ouverture s'effectue correctement avant de poursuivre le programme (nombreuses possibilités d'erreur : fichier effacé, erreur de nom, pas de droits de lecture,...). On utilise une instruction de test spécifique pour vérifier que l'ouverture du fichier s'est correctement effectuée, de type `try...catch...` (essaye ... sinon ...) permettant de prévoir une action de secours lorsqu'une une opération "risquée" échoue.

```
try{
    f = new FileReader ("/chemin/vers/le/fichier");
    g = new BufferedReader(f);
    System.out.println("L'ouverture s'est bien passée!");
}
catch (Exception e){
    System.out.println("Problème d'ouverture!");
    }
```

Créez un projet Java, recopiez les lignes ci-dessus en remplaçant `"/chemin/vers/le/fichier"` par l'emplacement réel de ventes.csv, et vérifiez que tout se passe bien...

### 2.2 Lecture d'une ligne

Comme nous l'avons vu dans l'introduction, le fichier `ventes.csv` est organisé sous forme de tableau où chaque ligne contient une liste de valeurs. Cette liste de valeurs est couramment appelée un **tuple**. Chaque ligne du fichier contient un tuple à l'exception de la première qui contient la liste des attributs, c'est à dire le **schéma des données**.

#### Récupération de la liste des attributs :

La première ligne du fichier contient le schéma de données. On remarque que les différents attributs sont séparés par des virgules (la virgule est donc le caractère de séparation.) On souhaite afficher

cette liste d'attributs.

La commande:

```
String s = null;
try{
    s = g.readLine();
}
catch (Exception e){
    System.out.println("Problème de lecture!");
    }
```

lit une ligne du fichier et la copie dans la variable `s`, de type "chaîne de caractères" (String). Appliquez la commande `readLine` et affichez `s`.

Découpage d'une chaîne de caractères

Une chaîne de caractères peut être découpée en plusieurs "morceaux", chaque morceau contenant une partie de la chaîne initiale. On utilise en général la commande `split` qui prend en paramètre le caractère séparateur à utiliser pour effectuer le découpage. Par exemple, la commande `s.split(",")` utilise le caractère séparateur `","`, et retourne une liste de chaînes de caractères.

On souhaite ici découper `s` pour construire un tableau de chaînes de caractères, appelé `tab1`, où chaque case du tableau contiendra le nom d'un attribut.

A faire :

- Définissez un tableau de chaînes de caractères `tab1`.
- `tab1` reçoit ensuite le résultat de la commande `split` appliquée sur la chaîne `s`.
- Créez un entier `m` qui reçoit le nombre d'attributs (`tab1.length`),
- puis affichez les attributs un par un (avec une boucle `for`).

#### 2.3 Extraction d'un tuple :

L'objet g est un flux de données. Lorsqu'une donnée a été lue, l'objet positionne sa tête de lecture sur la donnée suivante. Ainsi chaque nouvel appel à la méthode `readLine()` permet de lire une nouvelle ligne (jusqu'à ce qu'on atteigne la fin du fichier).

Ici, les lignes suivantes contiennent des valeurs. Commençons par lire la deuxième ligne.

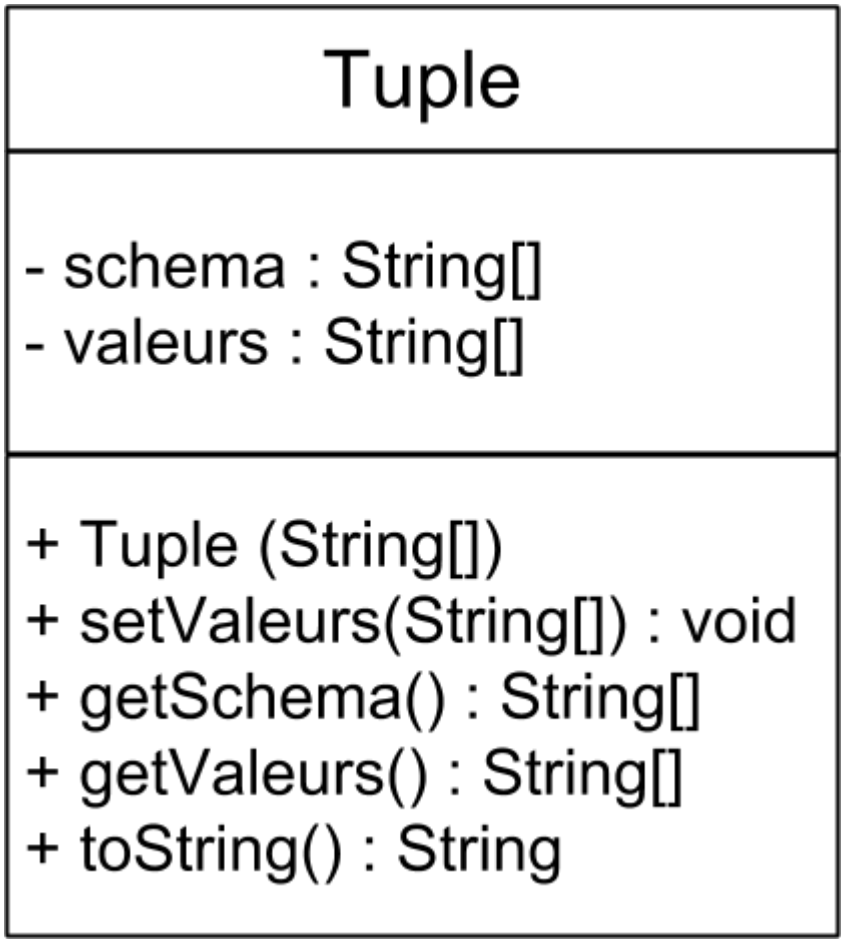
- Lisez une nouvelle ligne et affichez-la.
- Extrayez les différentes valeurs dans un tableau nommé tab2.
- Affichez ce tableau élément par élément
- En utilisant `tab1` et `tab2`, affichez les éléments sous la forme :

```
ATTRIBUT : valeur
```

## 3. Classe Tuple

Nous allons maintenant créer une classe pour gérer dans un même objet `Tuple` les données de type

schéma et valeurs.




Définissez dans votre projet cette nouvelle classe.

- Le constructeur initialise l'attribut `schema` avec le tableau fourni en paramètre, et initialise valeurs avec un tableau de même dimension tel que chaque case contient la valeur `null`.
- `setValeurs` permet de remplir le tableau valeurs avec le tableau fourni en paramètre. Pensez à vérifier la compatibilité de dimension…
- `getSchema` et `getValeurs` permettent de regarder le contenu de schema et valeurs
- `toString` permet d'afficher le contenu du tuple sous la forme :

```
ATTRIBUT1 : valeur1
ATTRIBUT2 : valeur2
etc...
```

Reprenez le programme principal de la question 2 en utilisant maintenant la classe `Tuple` pour stocker et afficher les données lues dans la troisième ligne du fichier.

- déclarer un objet `t` de type Tuple
- initialiser `t` avec le tableau `tab1`
- lire une nouvelle ligne et la découper, puis écrire les valeurs dans `t` à l'aide de la commande `setValeurs`
- afficher `t` :

```
system.out.println(t);
```

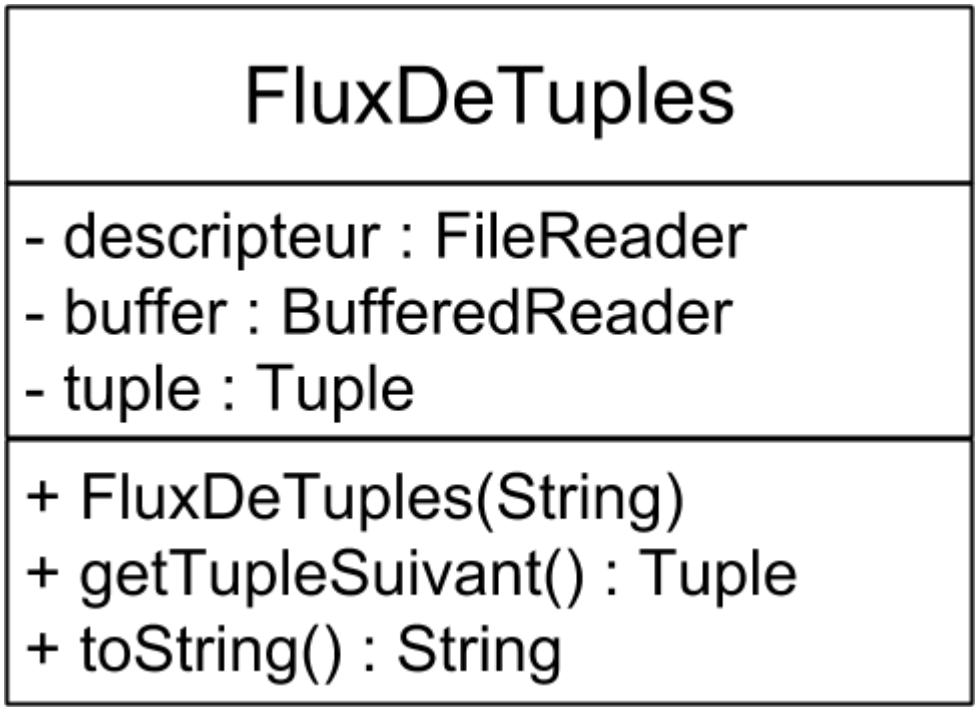
(doit produire le même type d'affichage que dans la question précédente)

Une fois ces manipulations effectuées, fermez le fichier avec la commande :

```
try {
    f.close();
}
catch (Exception e){
    System.out.println("Problème de fermeture!");
    }
```

## 4. Encapsulation des opérations d'accès aux données

Comme nous l'avons vu, les opérations d'accès aux données nécessitent d'effectuer des tests `try..catch` qui garantissent un traitement efficace des erreurs d'accès. Pour "alléger" le programme principal, nous allons créer une classe `FluxDeTuples` qui sert à "cacher" ces opérations compliquées...




Ajoutez la classe `FluxDeTuples` à votre projet.

Pour pouvoir fournir ces services, l'objet contient un champ descripteur de type `FileReader` et un champ buffer de type `BufferedReader`. Le tuple courant est stocké dans le champ `tuple`.

PS : N'oubliez pas d'inclure la librairie des entrées/sorties:

```
import java.io.*;
```

Cette classe propose les opérations suivantes:

- le constructeur initialise le `FileReader` et le `BufferedReader` à partir de l'emplacement du fichier fourni en paramètre, charge le schéma à partir de la première ligne du fichier, et

initialise tuple avec le schéma. Si le fichier ne s'ouvre pas correctement, on génère un message d'erreur (grâce à `try...catch`)
- La méthode `getTupleSuivant()` lit une nouvelle ligne, la découpe, met à jour le tuple courant et le retourne. Si la lecture ne s'effectue pas correctement, un message d'erreur est affiché (grâce à `try..catch`), et on retourne `null`.
- `toString()` affiche le tuple courant.

On peut éventuellement ajouter:

- `getTuple()` qui retourne le tuple courant

## 5. Extraction de la totalité du fichier

Testons maintenant notre classe `FluxDeTuples`. Il est possible de lire le fichier dans sa totalité en plaçant `h.litTupleSuivant()` dans une boucle. Dans le programme principal, ouvrez de nouveau le fichier (en initialisant le `FluxDeTuples h`),

```
FluxDeTuples h = new FluxDeTuples("/chemin/vers/le/fichier");
```

et exécutez les lignes suivantes :

```
while (h.litTupleSuivant()!=null){
   System.out.println("*****************************");
   System.out.println(h);
}
```

Même si vous avez correctement programmé la méthode `setValeurs`, Il doit y avoir un souci. Essayez de comprendre le problème. Le problème vient-il du fichier ou du programme?

Calculez (et affichez) le nombre de tuples incorrects.

## 6. Traitement par ''opencsv''

Pour lire correctement les fichiers csv, nous ferons appel à une librairie spécialisée Librairie `opencsv` : Récupérer et décompresser l'archive suivante:
http://sourceforge.net/projects/opencsv/files/opencsv/2.3/opencsv-2.3-src-with-libs.tar.gz

Extraire le fichier `opencsv-2.3.jar` (Java Librairy) situé dans le dossier `deploy`. Par défaut, Eclipse ne connaît pas cette librairie, il ne connaît que les librairies standard. Il faut donc ajouter cette nouvelle librairie au "chemin d'accès" du compilateur.

Pour modifier le chemin d'accès, allez dans :

```
project →properties →Java Build Path →  librairies → Add external jars
```

et ajoutez le fichier indiqué.

il ne reste plus qu'à l'importer dans la classe `FluxDeTuples`:

```
import au.com.bytecode.opencsv.CSVReader;
```

Le `CSVReader` remplace le `BufferedReader` utilisé précédemment.

```
lecteurCSV = new CSVreader(descripteur);
```

Grâce au `CSVreader`, l'opération de lecture retourne directement chaque ligne du fichier sous la forme d'un tableau de valeurs, sans faire (a priori) d'erreur de découpage.

Pour lire un tableau de valeurs, on utilise la commande `readNext()`

```
valeurs = lecteurCSV.readNext();
```

Modifiez la classe `FluxDeTuples` en utilisant le `lecteurCSV` au lieu du buffer sans modifier le programme principal. Si tout se passe bien, l'erreur précédente doit avoir disparu. Affichez le nombre de tuples contenu dans le fichier.