Pendant ce TP, vous allez continuer sur la couche Accès/Fichiers, pour finir le travail sur la logique d'insertion et sélection.
Vous allez notamment travailler sur la classe **HeapFile** et une nouvelle classe appelée **FileManager**.
En utilisant l'API du **FileManager,** vous allez également pouvoir coder la gestion des *commandes d'insertion et sélection*, et ainsi voir votre application tourner sur les premiers scénarios de test !

Pour pouvoir paralléliser le travail décrit ci-dessous, il est conseillé de coder rapidement des méthodes vides pour « l'API » de la classe **HeapFile** et de la classe **FileManager.**

A. Code : « API » de la classe HeapFile

Rajoutez sur la classe **HeapFile** les méthodes décrites ci-dessous.
Il s'agit des méthodes « publiques » nécessaires à l'insertion et à la sélection des records, qui seront appelées par la classe **FileManager**.
Pour coder ces méthodes, vous allez utiliser les « briques bas-niveau » codées lors du TP précédent.

- Insertion d'un Record :
  rid **InsertRecord (***record***)**, avec *record* un **Record** et *rid* un **Rid**.

- Lister tous les records dans le Heap File :
  listeDeRecords **GetAllRecords ()**, avec *listeDeRecords* une liste ou tableau de **Record**.

B. Code : la classe FileManager

Créez une classe **FileManager** *qui comportera une seule et unique instance*, appelée dans la suite *le* **FileManager**.
La seule variable membre de cette classe sera une liste ou tableau de **HeapFile**, appelée *heapFiles*,

Rajoutez dans la classe **FileManager** les méthodes suivantes, qui forment « l'API » de cette classe :

- Initialisation :
  Init ()
  Cette méthode doit :
  - parcourir la liste des **RelDef** de la **DBDef**
  - créer pour chaque telle **RelDef** un objet de type **HeapFile** en lui attribuant la **RelDef** en question
  - rajouter le **HeapFile** ainsi créé à la liste *heapFiles.*

  Rajoutez un appel à cette méthode dans la méthode **Init** du **DBManager**, *après l'appel à la méthode **Init** de la **DBDef** (pourquoi?).*

- Création d'une relation :
  CreateRelationFile (*relDef***)**,
  avec *relDef* une **RelDef.**
  Cette méthode doit :
  - créer un nouvel objet de type **HeapFile** et lui attribuer *relDef*
  - le rajouter à la liste *heapFiles*
  - puis appeler sur cet objet la méthode **createNewOnDisk**.

  Rajoutez un appel à **CreateRelationFile** dans la méthode **CreateRelation** du **DBManager**.

- Insertion d'un Record dans une relation :
  rid **InsertRecordInRelation (***record, relName***)**
  avec *record* un **Record,** *relName* une chaîne de caractères et *rid* un **Rid**.
  Cette méthode s'occupera de l'insertion de *record* dans la relation dont le nom est *relName*.
  Pour cela, il faudra parcourir la liste *heapFiles* pour trouver celui qui correspond à la relation en question, et ensuite appeler sa propre méthode **InsertRecord**.

- Lister tous les records d'une relation (« sélection sans conditions ») :
  listeDeRecords **SelectAllFromRelation (***relName***)**
  avec *listeDeRecords* une liste ou tableau de **Record** et *relName* une chaîne de caractères.
  Cette méthode doit retourner une liste contenant tous les records de la relation dont le nom est *relName.*

- Lister tous les records d'une relation qui respectent une condition sur une colonne :
  listeDeRecords **SelectFromRelation (***relName, idxCol, valeur***)**
  avec *listeDeRecords* une liste ou tableau de **Record**,
  relName une chaîne de caractères,
  idxCol un entier correspondant à un indice de colonne,
  et *valeur* une chaîne de caractères.
  Cette méthode doit retourner une liste contenant tous les records de la relation nommée *relName* pour lesquels la valeur sur la colonne *idxCol* (convertie en chaîne de caractères) est égale à *valeur.*
  Exemple : pour une relation R à deux colonnes (int, int)
  qui contient les trois Records (1,2), (3,4) et (1,4)
  SelectFromRelation(« R »,2, « 4 ») doit retourner { (3,4) , (1,4) }.
  SelectFromRelation(« R »,1, « 1 ») doit retourner { (1,2) , (1,4) }.

C. Code : Les commandes

Vous avez pour l'instant codé la gestion de la commande **create**, qui s'occupe de la création d'une relation.
En vous inspirant de la gestion de cette commande au niveau du **DBManager**, et en utilisant l'API du **FileManager**, vous allez pouvoir coder la gestion des *commandes d'insertion et sélection* décrites ci-dessous.
Vous allez également coder la gestion d'une commande très utile, appelée **clean**, qui « fait le ménage » au niveau de la DB.

Similairement à l'usage de **CreateRelation** pour la commande **create**, il est vivement conseillé de créer *une méthode dédiée à chaque commande* dans le **DBManager**.
Ces méthodes seront ensuite appelées depuis **ProcessCommand**.

C1. La commande clean

Rajoutez, dans votre application, la gestion de la commande **clean**.
Cette commande doit « faire un ménage général », autrement dit tout remettre dans un état qui correspond à un premier lancement de votre application. Plus particulièrement, il faudra :

- supprimer tous les fichiers .rf et le fichier Catalog.def
- « remettre tout à 0 » dans le **BufferManager**, **DBDef** et **FileManager**.
  Pour cela, il faut vider les listes, remettre tous les compteurs et les flags à 0, etc.
  Vous pouvez faire ces « remises à 0 » dans des méthodes **reset**, à créer sur chaque classe concernée, puis à appeler dans la méthode **clean** du **DBManager.**

C2. La commande insert

Rajoutez, dans votre application, la gestion de la commande **insert**.
Cette commande demande l'insertion d'un record dans une relation, en indiquant les valeurs (pour chaque colonne) du record et le nom de la relation.

Le format de cette commande est le suivant :
insert nomRelation val1 val2 … valn

On supposera que la relation existe, que le nombre de valeurs est correct, et que les valeurs correspondent bien aux types de colonnes de la relation.

C3. La commande insertall

Rajoutez, dans votre application, la gestion de la commande **insertall**.
Cette commande demande l'insertion de plusieurs records dans une relation.
Les valeurs des records sont dans un fichier csv : 1 record par ligne, avec la virgule comme séparateur.
On suppose que le fichier se trouve à la racine de votre dossier projet (au même niveau donc que les sous-répertoires Code et DB).

Le format de la commande est le suivant :
insertall nomRelation nomFichier.csv

C4. La commande selectall

Rajoutez, dans votre application, la gestion de la commande **selectall**.
Cette commande doit afficher tous les records d'une relation (1 record par ligne),
suivis par la phrase (sur une nouvelle ligne) *Total records = x* ,
où *x* est le nombre de records affichés.

Le format de la commande est le suivant :
selectall nomRelation

Pour afficher un record, on affiche ses valeurs séparées par des « ; » (espace, point virgule, espace)
Attention : veillez à respecter le format d'affichage demandé.

C5. La commande select

Rajoutez, dans votre application, la gestion de la commande **select.**
Cette commande doit afficher tous les records d'une relation qui ont une valeur donnée sur une colonne donnée.

Le format de la commande est le suivant :
selectall nomRelation indiceColonne valeur

L'affichage doit suivre le même format que celui demandé pour la commande **selectall.**

D. Information : Scénario de test

Vous trouverez ci-dessous un scénario de test de votre application qui comprend toutes les commandes codées lors de ce TP.

Le fichier S1.csv est disponible sur Moodle.

N'hésitez pas également à créer vos propres scénarios de test.
Et n'oubliez pas que les scénarios de test de l'application ne remplacent pas les tests à faire au niveau de chaque classe !

```
clean
create R 3 int string3 int
insert R 1 aab 2
insert R 2 abc 2
insert R 1 agh 1
selectall R                                  (résultat attendu : les trois tuples de R)
select R 1 1                                 (résultat attendu : deux tuples  (1 aab 2 et 1 agh 1) )
select R 3 1                                 (résultat attendu : un tuple  (1 agh 1) )
create S 8 string2 int string4 float string5 int int int
insertall S S1.csv
selectall S                                  (résultat attendu : 191 tuples)
select S 2 19                                (résultat attendu : 1 tuple)
select S 3 Nati                              (résultat attendu : 39 tuples)
```