Pendant ce TP, vous allez attaquer les deux premières couches de votre SGBD : la gestion de l'espace disque et la gestion des buffers. Vous allez donc coder le DiskManager et le BufferManager.

A. Information : mieux « paralléliser » votre travail

Pour bien travailler en parallèle, il faudra souvent que l'un de vous code « rapidement » des méthodes « vides » – qui puissent être par la suite appelées par le code de son collègue.
Par exemple, pour ce TP : pour pouvoir travailler sur le **BufferManager**, il faudrait dans l'idéal commencer par coder rapidement des méthodes « vides » au niveau du **DiskManager**.
Autrement dit, rendez d'abord disponible « votre API » !

B. Code : pageSize et constantes en général

Nous allons introduire progressivement plusieurs « constantes » (final...), *que vous pouvez regrouper dans une classe* **Constants**.
La première telle constante est la taille d'une page, *pageSize* **;** vous lui donnerez la valeur *4096*.

C. Code : gestion de l'espace disque

Votre SGBD *stockera chaque relation dans un fichier* (au sens OS).
Ces fichiers s'appelleront *Data_x.rf*, avec x entier >=0 l'identifiant du fichier : *Data_0.rf*, *Data_1.rf* etc.
Tous ces fichiers *seront placés dans votre sous-répertoire DB.*
Pour lire et écrire dans ces fichiers, vous allez utiliser des méthodes de lecture/écriture dans les fichiers binaires (*voir partie Documentation du TP1*).

C1. PageId

Chaque page dans chaque fichier sera identifiée par son PageId.
Le PageId est composé de deux parties :

- l'identifiant du fichier (un entier, le « x » dans Data_x)
- l'indice de la page dans le fichier (0 pour la première page, 1 pour la deuxième etc.).

Étant donné un PageId, nous pouvons ainsi retrouver sans ambiguïté le fichier et l'emplacement de la page dans ce fichier, donc le contenu de la page.

Créez une classe **PageId** avec deux variables membres : FileIdx et PageIdx.

C2. DiskManager

Créez une classe **DiskManager** *qui comportera une seule et unique instance.*

Rajoutez-y (au moins) les méthodes suivantes, qui forment « l'API » du gestionnaire disque :

- **CreateFile** (*fileIdx*), avec *fileIdx* un entier correspondant à un identifiant / indice de fichier.
  Cette méthode crée (dans le sous-dossier DB) un fichier Data_fileIdx.rf initialement vide.

- *pageId* **AddPage** (*fileIdx*), avec *fileIdx* un entier correspondant à un identifiant de fichier et *pageId* un **PageId**.
  Cette méthode rajoute une page au fichier spécifié par *fileIdx* (c'est à dire, elle rajoute *pageSize* octets à la fin du fichier) *et retourne un PageId correspondant à la page nouvellement rajoutée !*
  - *À vous de comprendre comment remplir ce PageId !*

- **ReadPage** (*pageId, buff*) avec *pageId* un identifiant de page et *buff* un buffer (utiliser le type choisi lors du TP précédent : byte[], ByteBuffer....).
  Cette méthode doit remplir l'argument *buff* avec le contenu disque de la page identifiée par l'argument *pageId*.
  Attention : c'est l'appelant de cette méthode qui crée et fournit le buffer à remplir!

- **WritePage** (*pageId, buff*) qui écrit le contenu de l'argument *buff* dans le fichier et à la position indiqués par l'argument *pageId.*

D. Code : gestion des buffers

Comme pour le **DiskManager**, vous allez devoir coder l'API que le **BufferManager** offre aux couches plus hautes, mais aussi sa gestion interne.

N'oubliez pas les éléments associés à une case (frame) :

- le buffer
- le PageId de la page qui s'y trouve chargée (si la case n'est pas libre)
- le pin_count
- le flag dirty

  ainsi que potentiellement des informations spécifiques à la politique de remplacement.

Pour la politique de remplacement, vous avez le choix entre LRU et Clock.

Il est conseillé (mais pas obligatoire) de créer une classe Frame.

D1. Constante frameCount

Le buffer pool contiendra *frameCount* cases (frames).

Vous allez définir *frameCount* comme une constante dans votre code et lui attribuer la valeur ***2*.**

D2. BufferManager

Créez une classe **BufferManager** *qui comportera une seule et unique instance*.
Rajoutez-y (au moins) les méthodes suivantes :

- *buff* **GetPage** (*pageId*) avec *pageId* un **PageId** et *buff* un buffer.
  Cette méthode doit répondre à une demande de page venant des couches plus hautes, et donc *retourner un des buffers associés à une case*.
  Le buffer sera rempli avec le contenu de la page désignée par l'argument *pageId*.
  *Attention : ne pas créer de buffer supplémentaire, « récupérer » simplement celui qui correspond à la bonne frame, après l'avoir rempli si besoin par un appel au **DiskManager**.*
  Attention aussi : cette méthode devra *s'occuper du remplacement d'une frame si besoin (donc politique de remplacement).*

- **FreePage** (*pageId, valdirty*) avec *pageId* un **PageId** et *valdirty* un entier ou booléen.
  Cette méthode devra décrémenter le pin_count et actualiser le flag dirty de la page.

- **FlushAll**(). Cette méthode s'occupe de :
  - l'écriture de toutes les pages dont le flag dirty = 1 sur disque
  - la remise à 0 de tous les flags/informations et contenus des buffers (buffer pool « vide »)

  Rajoutez un appel à cette méthode dans la méthode **Finish** du **DBManager.**