L'objectif est de manipuler des liste d'entiers de **manière récursive**. C'est à dire de réfléchir à chaque problème en le décomposant en problèmes identiques qui s'appliquerait sur une ou des listes plus petites.

Rappels rapide du cadre

Vous reprendrez les deux classes déjà vues pour modéliser les structures des listes :

```
public class Cellule {
        private int valeur;
        private Cellule suivante;
        /* A COMPLETER ... */
}
public class ListeDEntiers {
        private Cellule premier;
        /* A COMPLETER ... */
}
```

1. Rappelez comment on représente la liste vide (qui ne stocke pas d'entiers)
2. Une cellule déclare contenir une autre cellule en son sein, rappelez pourquoi cette définition n'est pas absurde ? Ou bien l'est-elle ?
3. Écrivez un constructeur pour la classe `Cellule` prenant seulement un argument entier
4. Écrivez un constructeur pour la classe `Cellule` qui prend en argument un entier et une cellule
5. Écrivez un constructeur pour la classe `ListeDEntiers` qui construit la liste vide
6. Écrivez une méthode `void add(int x)` pour la classe `ListeDEntier`, qui ajoute en tête de liste
7. Les attributs étant privés, implémentez des accesseurs pour la classe `Cellule`

Ecriture des méthodes classiques, récursivement

Il est assez normal que vous n'ayez pas toujours le résultat voulu du premier coup, alors n'hésitez pas à faire quelques schémas puis de vérifier en faisant des tests pour des cas limites. Après plusieurs relectures et ajustements vous devriez arriver à une formulation dont vous serez satisfait.
Même lorsque cela n'est pas explicitement demandé, il vous faudra souvent créer des méthodes supplémentaires nécessaires pour mettre en place le cadre du raisonnement récursif. La plupart du temps vous pouvez faire comme en cours : une méthode sera initiée dans la classe des listes, et la récursivité déléguée à une autre méthode dans la classe des cellules.

1. Écrivez une méthode `description` qui renvoie une chaîne de caractères décrivant la liste d'entiers séparés par des espaces, ou "la liste est vide" si c'est le cas.
2. Reprenez votre code pour qu'il retourne un résultat sous une forme plus élégante. Par exemple : `"(1;5;4)"`
3. Écrivez une méthode `taille ()`, sans argument, qui renvoie le nombre d'éléments de la liste. Écrivez également une méthode `somme()`, qui renvoie la somme des éléments de la liste.
4. Écrivez une méthode `ajouter_en_i` qui prend deux arguments entiers `i` et `v` et ajoute l'entier `v` en position `i` si elle existe, sinon elle l'ajoute en interprétant cet indice "au plus près" de ce qu'il peut signifier.
5. Écrivez une méthode récursive `boolean supprimer_en_i` qui supprime le `i`-ème élément de la liste.

6. Écrivez une méthode `egal(ListeDEntiers arg)` qui teste l'égalité de deux listes simplement chaînées (à savoir `arg` et `this`). Deux listes sont égales si elles contiennent les mêmes valeurs dans le même ordre.
7. Écrivez une méthode `int supprimer_k_premieres_occ(int k, int v)` qui supprimera les `k` premières occurrences d'un entier `v`. Elle retournera le nombre de valeurs effectivement supprimées.
8. On souhaite écrire une méthode `swap` qui échange deux entiers de la liste, situés aux positions i et j données en arguments. Ecrivez ce qui est nécessaire à sa réalisation. (Laissez les cellules là où elles sont, ne permutez que leurs contenus)
9. Écrivez une méthode `obtenir_sous_liste_inf_k (int k)` qui renvoie une nouvelle liste, construite à partir de la liste courante en y copiant les entiers strictement inférieurs à `k`. La liste d'origine n'est pas modifiée, on travaillera sur des nouvelles cellules. L'ordre d'apparition est respecté.

(Facultatif) Pour compléter votre entrainement

Écrivez les méthodes précédentes sous forme itérative.