Seul document autorisé : une feuille A4 recto-verso manuscrite. Aucune machine. Le barème est indicatif.
Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.

Tout au long de l'énoncé, vous pouvez introduire des méthodes supplémentaires si vous en avez besoin. Les méthodes mentionnées dans les énoncés des exercices font référence à celles apparaissant dans les interfaces décrites par l'énoncé.

1 Listes chaînées

L'utilisation de collections de l'API Java est interdite dans cette partie (entre autres `LinkedList` et `ArrayList`).

Un service de réservation de trains cherche à modéliser son système. Un utilisateur souhaitant réserver une place ne recevra pas un numéro de place, mais uniquement un numéro de wagon, le placement à l'intérieur d'un wagon étant libre. Pour modéliser ce système, on décide de représenter un train par une liste chaînée, chaque wagon étant une cellule de cette liste. On utilisera donc les classes : **`Train`** pour la tête de liste et **`Wagon`** pour chaque cellule. Par ailleurs chaque wagon possèdera un identifiant entier individuel unique (l'unicité porte sur tous les wagons, indépendamment du train auquel le wagon est rattaché), un entier donnant sa capacité (c'est-à-dire le nombre de places dans le wagon, réservées ou non) et un entier donnant le nombre de places non encore réservées dans le wagon.

L'interface de la classe **`Train`** devra contenir (au minimum) les méthodes suivantes :

```
/** classe Train */
/** affiche la composition du train */
int afficherComposition();
/** supprime toutes les reservations du train */
void vider();
/** renvoie le nombre total de places dans le train (reservees ou non) */
int capaciteTotale();
/** attribuer des places a une colonie de vacances */
int[][] placerColo(int nbEnfants, int nbAdultes);
```

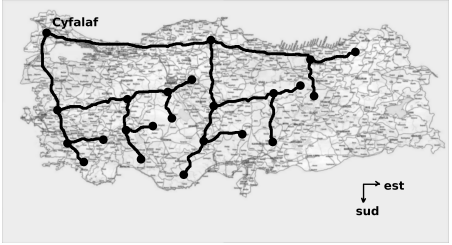
Tous les attributs des classes `Train` et `Wagon` doivent être privés.

Exercice 1. (1,5 points) Donner les attributs et les constructeurs des classes **`Train`** et **`Wagon`**. Les constructeurs doivent permettre de construire un train, vide ou non, et d'écrire les autres méthodes.

Exercice 2. (2,5 points) Écrire les méthodes **`afficherComposition`** (qui pour chaque wagon affiche son identifiant et sa capacité), **`vider`** et **`capaciteTotale`**.

Exercice 3. (2,5 points) Écrire la méthode **`placerColo`**, sachant qu'un groupe d'enfants ne peut se trouver seul dans un wagon. Le principe de cette méthode est le suivant : elle reçoit en argument le nombre d'enfants et le nombre d'adultes à placer ; l'algorithme à suivre (il ne trouve pas nécessairement une solution, mais c'est celui qu'on vous demande) est le suivant : dans le premier wagon, on regarde le nombre maximal d'enfants qu'on peut placer avec un adulte, de même dans le deuxième wagon avec les enfants et adultes restant à placer, etc. jusqu'à ce que tout le monde ait une place (on ne place pas un adulte tout seul dans un wagon s'il reste des enfants à placer ; s'il ne reste plus d'enfants à placer, on place les adultes dès que possible). Si le remplissage est impossible de cette manière, la réservation ne se fait pas et on renvoie **`null`**. Si le remplissage est possible, on met à jour les attributs correspondants dans chaque wagon et on renvoie un tableau bidimensionnel d'entiers, de longueur le nombre de wagons du train et de hauteur 3 : le premier entier de la colonne **`i`** est l'identifiant du wagon, le deuxième correspond au nombre de places réservées pour des enfants dans le wagon et le troisième au nombre de places réservées pour des adultes dans le wagon, l'entier **`i`** correspondant à l'ordre d'apparition des wagons dans le train, c'est-à-dire dans la liste chaînée (et non à leur identifiant).

2 Arbres binaires



Le pays de Petryal est rectangulaire, sa capitale Cyfalaf se situant dans le coin nord-ouest du pays. Les autorités décident de rénover leur réseau ferré qui a été construit "à la SNCF" : toutes les voies partent de, ou finissent dans la capitale. Une gare est *terminale* si c'est un cul-de-sac, c'est-à-dire qu'on ne peut y arriver que par un unique tronçon, et dont on ne peut repartir que par ce même tronçon pris en sens inverse. Par esprit de simplification, le réseau a toujours été étendu à partir d'une gare terminale : on a toujours construit nécessairement deux tronçons de voie vers deux autres gares, une plutôt

à l'est, l'autre plutôt au sud, en évitant tout quadrillage. En dehors de la capitale, chaque gare est sur le trajet d'une unique ligne (dans le sens aller et le sens retour). Un exemple est donné au dos.

On veut modéliser ce réseau de chemin de fer centralisé par un arbre binaire, ayant pour racine la capitale. La classe **`Gare`** servira à représenter les nœuds de l'arbre et la classe **`ReseauFerre`** stockera une référence vers la racine de cet arbre. Un nœud ne contient pas de référence vers son père. Les gares terminales sont donc celles qui sont représentées par des nœuds sans fils (c'est-à-dire des feuilles). Les autres gares ont toutes deux fils.

L'interface de la classe **`ReseauFerre`** devra contenir (au minimum) les méthodes suivantes :

```
/** classe ReseauFerre */
/** renvoie la gare atteinte en suivant le chemin donne en argument */
Gare gareAtteinte(String chemin);
/** renvoie le prix de renovation de toutes les gares */
int prixRenovationTotale(int prixMCarre);
/** renvoie la gare la plus grande en surface */
Gare laPlusGrande();
/** renvoie la gare a renover en priorite */
Gare gareARenover(int somme, int prixMCarre);
```

Tous les attributs des classes `ReseauFerre` et `Gare` doivent être privés.

Exercice 4. (1,5 points) En plus de ce qui a été indiqué dans l'introduction de cette partie, chaque instance de la classe **`Gare`** contient un entier qui est l'année de la dernière rénovation de la gare (ou l'année de création si aucune rénovation n'a été faite depuis) et un autre représentant la surface de la gare. Donner la liste des attributs des deux classes (on ne demande pas de constructeur).

Exercice 5. (2 points) On veut obtenir une gare grâce à la description du chemin à partir de la capitale. Ce chemin est supposé donné sous forme d'une chaîne de caractères ne contenant que des **`'e'`** (pour *est*) et des **`'s'`** (pour *sud*). Écrire la méthode **`gareAtteinte`**. Attention à bien gérer le cas où l'argument n'est pas au bon format et celui où la gare n'existe pas. Pour rappel, la classe **`java.lang.String`** contient une méthode **`char charAt(int index)`** qui prend en argument un indice et renvoie le caractère qui se situe à cet indice (le premier caractère ayant 0 pour indice).

Exercice 6. (2 points) Ecrire la méthode **`prixRenovationTotale`** qui prend en argument le prix de rénovation du m^2 de gare et renvoie le prix de rénovation de toutes les gares du réseau.

Exercice 7. (2 points) Ecrire la méthode **`laPlusGrande`**. S'il y a plusieurs gares qui répondent au critère, en renvoyer une seule.

Une gare est dite *plus ancienne* qu'une autre si sa date de rénovation est antérieure (voir exercice 4).

Exercice 8. (2 points) Ecrire la méthode **`gareARenover`** qui prend en argument une somme et un prix au m^2 (supposés entiers) et renvoie la plus ancienne gare pouvant être rénovée avec cette somme, en supposant que le prix est le prix de la rénovation d'un m^2. S'il y a plusieurs gares qui répondent au critère, en renvoyer une seule.

Exercice 9. (4 points) Proposer de nouvelles structures de données pour **`ReseauFerre`** et **`Gare`** qui permettraient d'avoir un arbre doublement chaîné (chaque nœud contiendrait une référence vers son père, la racine n'ayant pas de père) —appelez-les **`ResD`** et **`GarD`**. Ecrire un constructeur de **`ResD`** qui prend en argument une instance de **`ReseauFerre`** (supposée non **`null`**, sans avoir à le vérifier) et construit une instance de **`ResD`** qui représente le même réseau ferré.