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 nettoyage de trains cherche à modéliser le travail de ses employés. Pour cela, il 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 train possèdera un identifiant entier individuel unique et chaque wagon un booléen permettant de savoir s'il est propre ou non.

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

```
/** classe Train */
/** renvoie l'identifiant du train */
int id();
/** renvoie le nombre de wagons du train */
int longueur();
/** passe tous les wagons a ``sale'' */
void estToutSale();
/** renvoie le nombre de wagons sales */
int wagonsANettoyer();
```

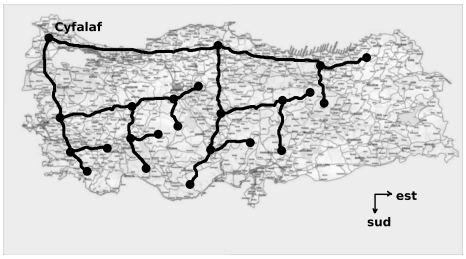
Tous les attributs des classes `Train` et `Wagon` doivent être privés.

Exercice 1. (2,5 points) Donner les attributs et les constructeurs des classes **`Train`** et **`Wagon`**, ainsi que la méthode **`id`**. Les constructeurs doivent permettre de construire un train, vide ou non.

Exercice 2. (3,5 points) Écrire les méthodes **`longueur`**, **`estToutSale`** et **`wagonsANettoyer`**.

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 mettre en place un réseau ferré "à la SNCF" : toutes les voies partent de ou finissent dans la capitale. Une gare est *terminale* si c'est un cul-de-sac : une seule façon d'y arriver et on repart en reprenant le même chemin dans le sens inverse. Le trajet d'un train se fait toujours entre la capitale et une gare terminale (dans un sens ou dans l'autre), avec arrêt à chaque gare intermédiaire. Par esprit de simplification, quand on décide d'étendre le réseau à partir d'une gare terminale, on 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 ci-dessus.



On veut modéliser ce réseau de chemin de fer centralisé par un arbre binaire, de 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 la plus a l'est */
Gare aLEstToute();
/** affiche la liste des noms de toutes les gares du reseau */
void lsGares();
/** renvoie la liste des gares entre la capitale et l'argument, sur le chemin le plus court
```

```
(dans l'ordre) */
ArrayList<Gare> trajetVers(Gare gare);
/** renvoie la liste des gares entre les deux arguments, sur le chemin le plus court
(dans l'ordre) */
ArrayList<Gare> trajetEntre(Gare depart, Gare arrivee);
/** renvoie le temps de trajet entre deux gares */
int tempsTrajet(Gare depart, Gare arrivee);
```

Tous les attributs des classes `ReseauFerre` et `Gare` doivent être privés. Une partie de l'interface de la classe **`ArrayList`** est donnée à la fin du document, vous pouvez utiliser des méthodes de cette classe qui ne sont pas rappelées.

Exercice 3. (2 points) En plus de ce qui a été indiqué dans l'introduction, chaque instance de la classe **`Gare`** contient une chaîne de caractères qui est le nom de la gare et un entier correspondant au temps (en minutes) mis par un train pour arriver à cette instance à partir de son père. Donner la liste des attributs des deux classes.

Exercice 4. (1 point) On veut connaître la gare la plus à l'est, c'est-à-dire celle obtenue en suivant à partir de la racine tous les fils est. Donner le code de la méthode **`aLEstToute`**.

Exercice 5. (2 points) Ecrire la méthode **`lsGares`**.

Exercice 6. (3 points) Ecrire la méthode **`trajetVers`**. Attention : les gares doivent être stockées dans le bon ordre et le chemin minimal (c'est-à-dire sans allers-retours entre deux gares). N'oubliez pas le cas où la gare fournie en argument n'existerait pas dans le réseau considéré.

Exercice 7. (3 points)

1. Écrire une fonction qui prend deux arguments de type **`ArrayList<Gare>`** et renvoie le plus grand entier n tel que les n premières cases des deux arguments coïncident.
2. Ecrire la méthode **`trajetEntre`**. Attention : les gares doivent être stockées dans le bon ordre et le chemin minimal (c'est-à-dire sans allers-retours entre deux gares). En cas de changement de train à une gare, cette dernière doit apparaître deux fois de suite dans la liste. On est amené à changer de train si la gare de départ et la gare d'arrivée ne se situent pas sur la même ligne. On peut commencer par récupérer les trajets entre la capitale et la gare de départ d'une part, et entre la capitale et la gare d'arrivée de l'autre et les recoller correctement, en supprimant l'éventuel tronçon commun.

Exercice 8. (3 points) Tous les trains circulent entre la capitale et une gare terminale, et s'arrêtent sur chaque gare du parcours pendant 2 minutes. Un train circule à la même vitesse dans les deux sens. En cas de changement de train à la gare de la capitale, il faut compter 15 minutes, dans une autre gare 10 minutes.

Ecrire la méthode **`tempsTrajet`**.

Une partie de l'interface de la classe **`java.util.ArrayList`** :

```
/** class ArrayList<E> */
/** Constructs an empty list with an initial capacity of ten. */
ArrayList();
/** Appends the specified element to the end of this list. */
boolean add(E e);
/** Inserts the specified element at the specified position in this list. */
void add(int index, E element);
/** Returns the element at the specified position in this list. */
E get(int index);
/** Returns true if this list contains no elements. */
boolean isEmpty();
/** Removes the element at the specified position in this list. */
E remove(int index);
/** Removes the first occurrence of the specified element from this list, if present. */
boolean remove(E e);
/** Returns the number of elements in this list. */
int size();
```