IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP11-15 – Trains et circuits

Fondements théoriques du développement des logiciels concurrents

AAGOUR Fouad
LAGSSAIBI Adam

Exercice 1 (Le comportement d'un train):

Question 1.1):

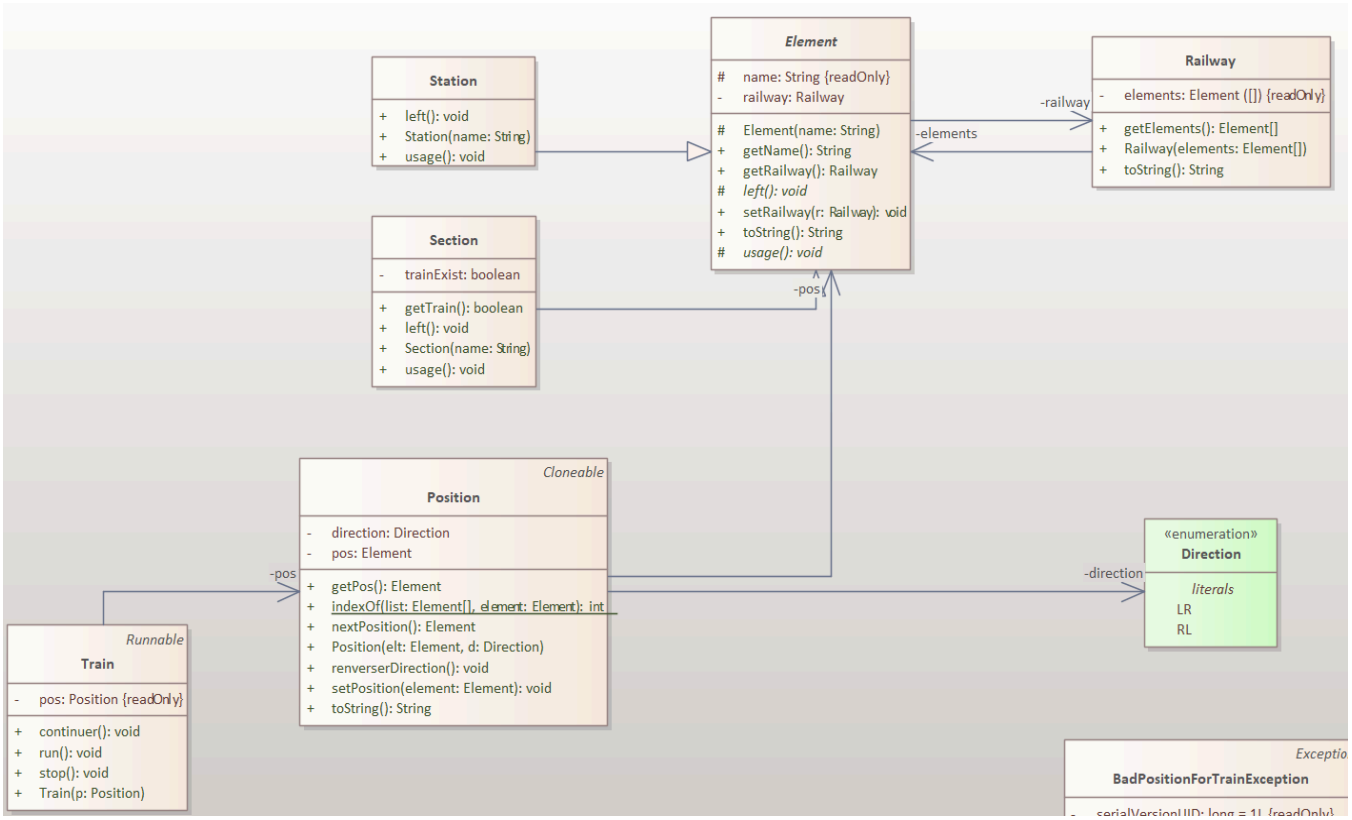
Dans le diagramme de classe fournit, les différentes classes ont des rôles spécifiques dans la réalisation du déplacement d'un train. Voici le rôle de chaque classe :

- **Element** : Cette classe abstraite représente les éléments de base du réseau ferroviaire, comme les sections et les stations.
- **Railway** : Cette classe représente le réseau ferroviaire dans son ensemble. Elle contient un tableau d'éléments qui composent le réseau.
- **Section** : Cette classe hérite de la classe Element et représente une section spécifique du réseau ferroviaire.
- **Station** : Cette classe hérite également de la classe Element et représente une station du réseau ferroviaire. Elle a une taille spécifique et est capable de stocker des trains.
- **Train** : Cette classe représente un train dans le système. Elle a un nom et une position, et elle peut se déplacer d'une position à une autre sur le Railway
- **Position** : Cette classe représente la position actuelle d'un train sur le réseau ferroviaire.
- **Direction** : Cette énumération définit les directions possibles dans lesquelles un train peut se déplacer : de gauche à droite (LR) ou de droite à gauche (RL).

Dans le cadre du déplacement d'un train : La classe Train est responsable de représenter les trains individuels, de stocker leur position actuelle et de fournir des méthodes pour les déplacer. La classe Position est utilisée pour représenter la position actuelle d'un train sur le réseau ferroviaire. Elle contient des informations sur l'élément spécifique (section ou station) où se trouve le train et la direction dans laquelle il se déplace. Les autres classes, telles que Element, Railway, Section et Station, fournissent la structure de base du réseau ferroviaire et des éléments qui le composent.

Question 1.2):

On en a ajouté plusieurs dans notre diagramme de classe. voilà des explications et quelques méthodes nécessaires: Dans la méthode **nextPosition()** de la classe Position, la prochaine position du train est calculée en fonction de sa direction et de sa position actuelle sur le réseau ferroviaire. Cette méthode utilise la méthode indexOf() pour trouver l'index de la position actuelle du train dans le tableau des éléments du réseau ferroviaire. Déplacement du train vers la prochaine position : La méthode **setPosition()** de la classe Position est utilisée pour déplacer le train vers la prochaine position calculée. Cette méthode est appelée dans la méthode **continuer()** de la classe Train. Interaction du train avec les éléments du réseau : Le train interagit avec les éléments du réseau ferroviaire en appelant les méthodes **usage()** et **left()** sur chaque élément. Ces méthodes sont définies dans les classes Station et Section qui étendent la classe abstraite Element. La méthode **usage()** simule l'utilisation de l'élément par le train (comme l'arrêt à une station ou le passage d'une section de voie). La méthode **left()** est appelée lorsque le train quitte un élément.

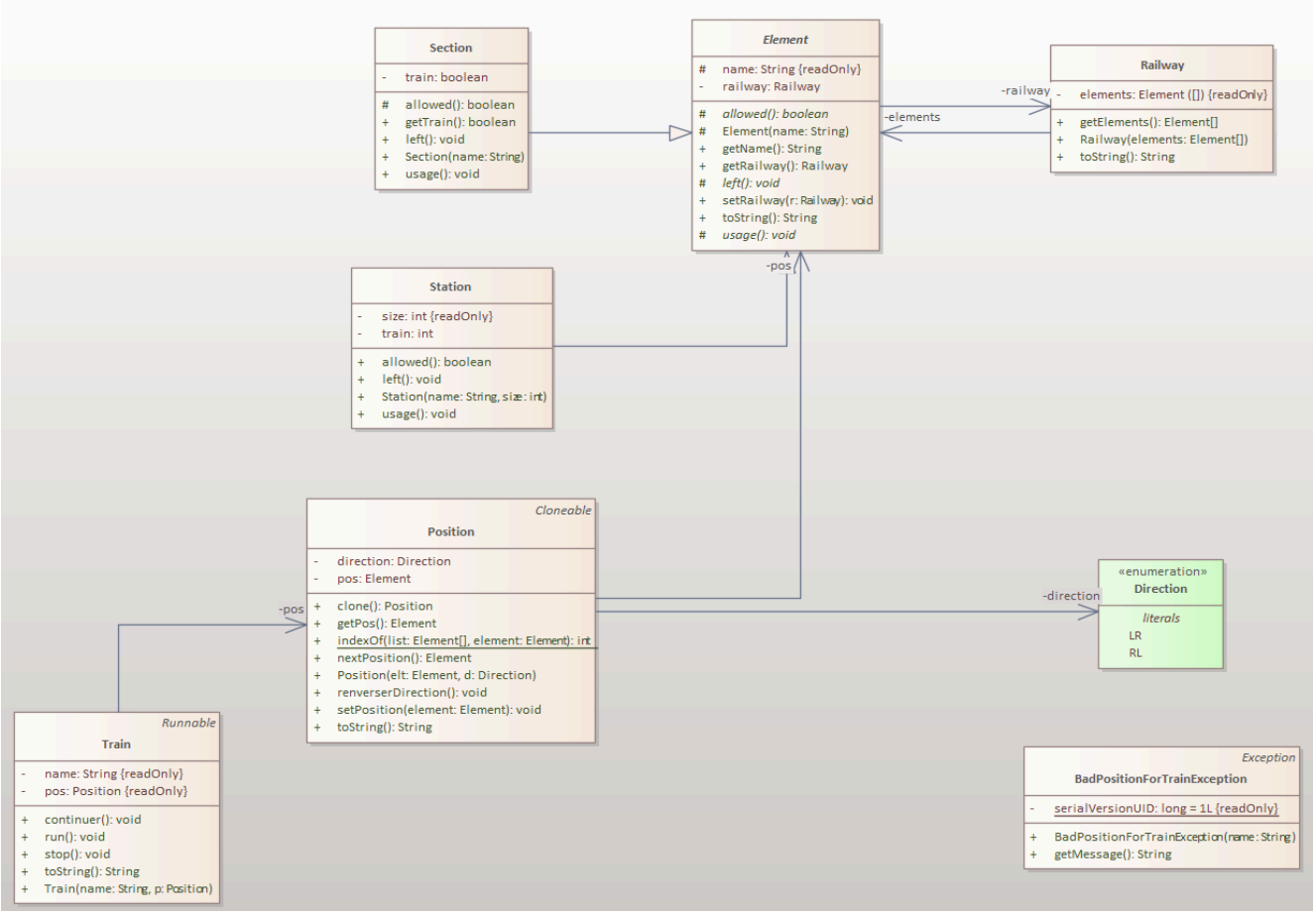


Question 1.3):
(voir le code premier package nommé train 0, vous pouvez tester la classe Main pour vérifier le bon fonctionnement de nos méthodes)

Exercice 2 (Plusieurs trains sur la ligne) :

Question 2.1):

La partie du code pour cet exercice se trouve dans le package nommé EX2 dans notre code. le diagramme de classe pour cette partie est le diagramme suivant:



Question 2.2):

Les variables qui permettent d'exprimer l'invariant de sûreté pour la ligne de trains sont :

- **train** dans la classe Section, qui indique s'il y a un train dans la section.
- ***size*** dans la classe Station: Nombre maximum de trains autorisés dans une gare
- **train** dans la classe Station, qui indique le nombre de trains actuellement dans chaque station.

Question 2.3):

- Pour chaque gare, le nombre de trains présents ***(train)*** est inférieur ou égal au nombre maximal de trains autorisés dans cette gare ***(size)***. (***Tout simplement train<size***)
- Pour chaque section, il y a au maximum un train présent. (**Ceci est vérifié par le fait de définir une variable train dans la classe Section sous forme de boolean càd soit un train existe dans la section ou bien il n' y a aucun train dans la section**)

Question 2.4):

Voici quelques-unes de ces actions, identifiées par le nom des fonctions utilisées dans le code fourni :

- **continuer()** : Cette fonction permet au train de continuer son mouvement vers la prochaine position sur le réseau ferroviaire. Elle est appelée lorsque le train quitte une gare ou une section de voie ferrée.
- **stop()** : Cette fonction permet au train de s'arrêter ou de changer de direction. Elle est appelée lorsqu'un train arrive à une gare, où il peut être amené à attendre avant de pouvoir continuer son trajet.
- **usage()** : Cette fonction simule l'utilisation d'une gare par le train. Elle peut représenter des actions telles que l'embarquement ou le débarquement des passagers, ainsi que d'autres opérations spécifiques à une gare.
- **left()** : Cette fonction indique que le train quitte une gare ou une section de voie ferrée. Elle peut impliquer des actions telles que la libération de la voie pour d'autres trains ou la mise à jour de l'état du train et de l'élément sur lequel il se trouvait.

Ces actions sont essentielles pour la simulation du système ferroviaire et sont implémentées dans les classes Train, Section et Station du code fourni.

Question 2.5):

Les actions critiques d'un train, telles que **continuer(), stop(), usage() et left()**, doivent être appelées dans la classe Train. C'est dans cette classe que le comportement du train est défini, y compris ses actions lorsqu'il se déplace sur le réseau ferroviaire, entre dans une gare, en sort, etc. usage() et left() sont définies dans la classe Element (des méthodes abstraites) et continuer() et stop() sont définies dans la classe Train.

Question 2.6):

Pour garantir une synchronisation correcte dans un environnement multithread, en plus des méthodes continuer(), stop(), usage() et left() déjà mentionnées, il est également nécessaire d'ajouter les méthodes suivantes :

- **wait()** : Cette méthode doit être appelée lorsque le train doit attendre dans une situation où il n'est pas autorisé à continuer, par exemple lorsqu'il arrive à une gare et tous les quais sont pleins. Cette méthode sera appelée dans la boucle run() de la classe Train.
- **notifyAll()** : Cette méthode doit être appelée lorsque la situation change et que le train est autorisé à reprendre son mouvement, par exemple lorsque le train précédent quitte une gare et libère un quai. Cette méthode doit être appelée après avoir mis à jour l'état qui autorise le train à continuer.

Ces méthodes doivent être correctement synchronisées pour garantir un fonctionnement sûr et cohérent du système.

Question 2.7): (voir le code package 2 nommé train 1)
Question 2.8): (voir le code package 2 nommé train 1)

Exercice 3 (Éviter les interblocages...):

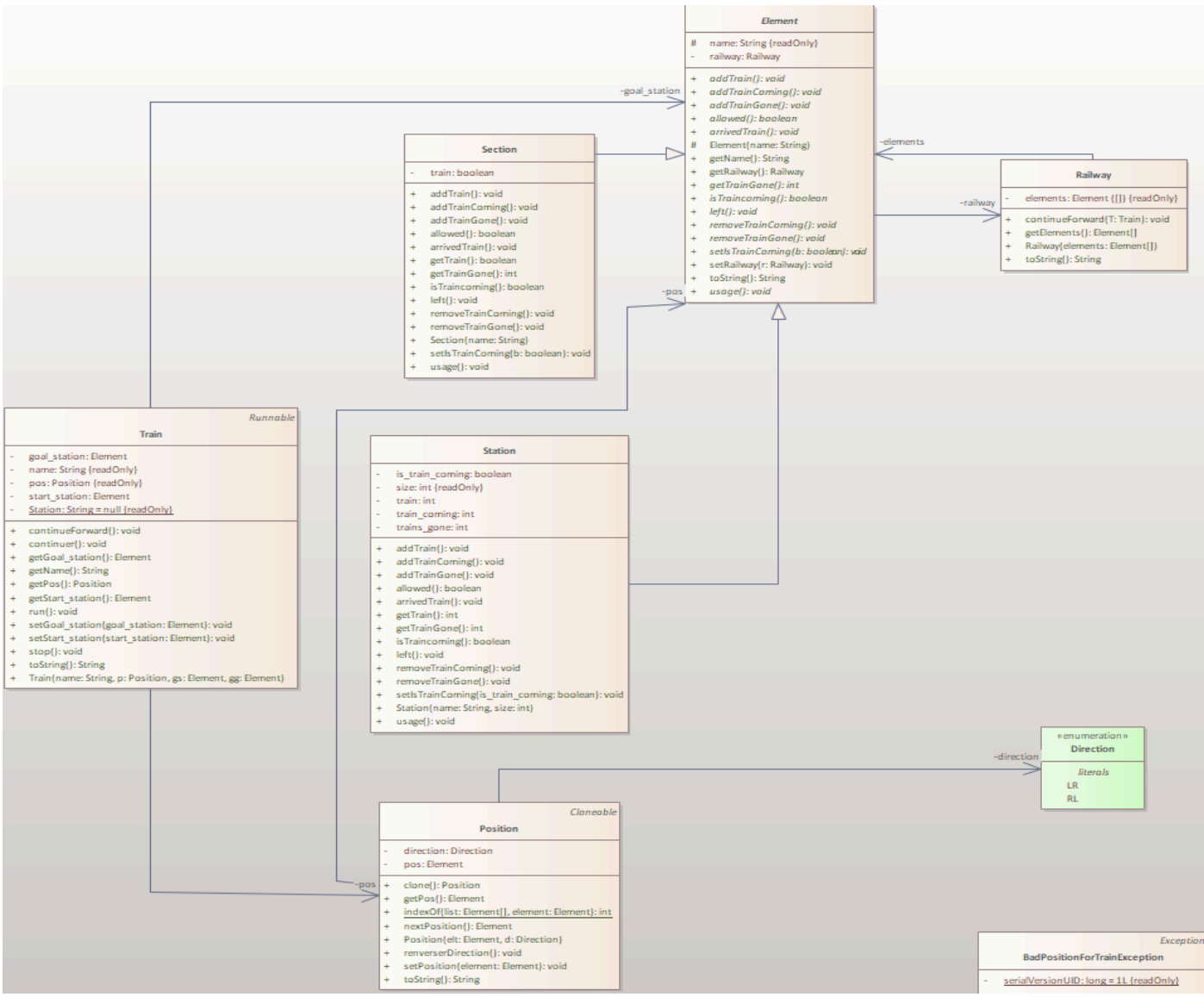
Question 3.1):

L'attribut direction de la classe Pos, qui est aussi l'attribut de la classe Train, permet de savoir la direction d'un train donné. Il faut d'abord s'assurer de savoir si un train est engagé et stocker son sens. Ensuite, il faut vérifier à chaque fois qu'une gare souhaite laisser partir un train. Elle doit s'assurer qu'il n'y a pas de train en ligne ou qu'ils sont dans le même sens. Normalement, ils ne devraient pas se chevaucher puisque nous avons déjà implémenté que le train ne peut entrer sur une ligne s'il y a déjà un train.

Pour le cas de deux gares, il suffit d'ajouter un attribut à la classe Station pour savoir si elle a le droit de lancer un train dans un certain sens (attribut de type **boolean**) nommé dans notre code `is_train_coming`, qui est modifié par l'autre station si elle lance un train dans un certain sens.

Question 3.2):

Si un train arrive dans la gare depuis la direction opposée, alors la gare ne peut pas laisser partir un train dans cette direction. Cela signifie que la variable **is_train_coming** doit être vérifiée avant de permettre le départ d'un train dans une direction donnée à partir d'une gare. Si **is_train_coming** est vrai pour une direction, cela signifie qu'un train est en route dans cette direction, et la gare ne doit pas permettre le départ d'un train dans cette direction pour éviter les interblocages.



Question 3.3):

La classe responsable de la gestion de ces variables est la classe **Station**.

Question 3.4):

voir notre code (package 3 nommé train2)

Question 3.5):

voir notre code (package 3 nommé train2).