Explication de la solution : Structure

Le programme final est structuré ainsi :

**La classe Train** représente l’objet principal exécutée par le programme. Elle implémente la classe Runnable, une version approfondie de la classe Thread, avec des fonctions préconçues pour simplifier leur manipulation.

Un Train a un String Nom pour afficher ses actions, une Voie voieTrain pour indiquer sur quelle Voie il circule, une Case Int pour indiquer sur quelle case il se trouve au lancement du programme.

Train a des méthodes qui servent à la faire avancer sur une Voie, et l’autre pour lancer son exécution en tant que Thread.

**La classe Voie** représente l’objet sur lequel interagit un Train, donc la Voie sur laquelle il circule.

Une Voie a une ArrayList de Cases qui représente sont trajet. Ce trajet a une longueur int trajetLongueur. Une voie est initialisée avec un nombre de Cases.

Voie a une méthode qui sert à remplir son trajet de Cases, et d’identifier certaines de ses cases comme Gares.

**La classe Case** représente la collection d’objets qui composent une Voie.

Une Case possède un sémaphore pour indiquer si elle est occupée, un Train trainNum pour indiquer quel Train occupe cette Case si elle est occupée, un entier selfCase et nextCase pour indiquer son identifiant de Case et un pointeur vers la prochaine Case, respectivement.

Gare1, 2 et 3 sont des Cases spécialisées, avec nextVoie1, 2 et 3 indiquant quelle est sa prochaine Case en fonction de la Voie vers laquelle le Train doit être placé.

Case a une fonction qui a pour but d’identifier quelles sont ses nextVoie, ainsi que les fonctions pour entrer et sortir une et d’une Case, avec des cas selon s’il s’agit d’une Gare et laquelle. Bien que « synchronized » n’est pas déclaré avant ces méthodes, l’équivalent a été obtenu grâce à des vérifications de sémaphores et de partage de données et d’informations.

**La classe Chrono** représente un chronomètre utilisé pour compter le temps écoulé depuis le lancement du programme.

**La classe TRAINMAIN** est la classe qui sert de lancement du programme.

Explication de la solution : Fonctionnement

Le programme final fonctionne ainsi : lors de son exécution, on démarre un Chronomètre. C’est grâce à celui-ci qu’on arrive à déterminer à quel moment les Trains tentent d’entrer et de quitter des Gares.

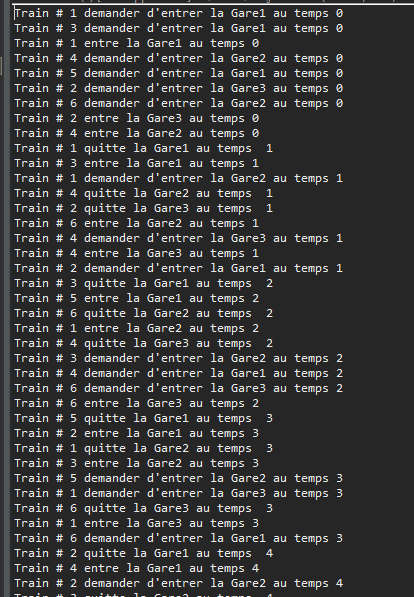
On instancie par la suite chacune des Voies sur lesquelles les Trains circulent, puis le positionnement des Gares sur celles-ci, car chacune possède de coordonnées uniques relatives aux Voies.

Ensuite, on crée les Trains en guise de Threads, en affixant du même coup leur Case de départ selon les Voies qu’ils occupent.

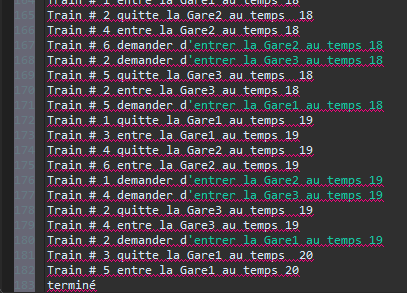
Afin d’assurer le bon fonctionnement des Threads, chaque Case et chaque Gare possède un sémaphore. Cela a pour effet de limiter le nombre de Trains qui peuvent y accéder, et donc d’éviter les collisions. Si une Case ou une Gare est occupée, donc que son Sémaphore a ses ressources épuisées, le Train demandeur est placé en fil d’attente, jusqu’à ce qu’une ressource se libère et qu’il s’agisse de son tour. On retire donc de chaque case occupée par un Train son sémaphore.

Puisque les Gares sont des cas particuliers de Cases, on doit leur attribuer un sémaphore unique et différent d’une Case normale, car l’adressage des Gares pour les entrées et sorties est plus complexe qu’une Case classique.

Enfin, on peut démarrer la simulation de la circulation.



Lors de l’exécution, des messages sont affichés de façon constante en console afin d’illustrer les mouvements des Trains. Il est possible d’observer le bon fonctionnement des sémaphores et des files d’attente, en raison des Trains qui sont incapables d’entrer dans des Gares occupées. Après le délai d’une seconde, on remarque que le prochain Train à accéder une Gare nouvellement libérée est le premier à avoir fait la demande d’y entrer lors de son occupation. On peut voir par la suite un enfilement et une rotation logique des accès en fonction de cette liste d’attente.



Ultimement, le programme cesse lorsque 60 entrées en Gares ont eu lieu. Les Threads prennent fin après leur dernière action en cours, ce qui explique pourquoi il existe un peu plus de 180 lignes en console : les Trains qui, au moment de la dernière entrée/sortie de Train étaient en processus de demande d’accès à une Gare, impriment leur dernière communication avec la console, avant que le programme soit enfin arrivé à terme.