I.U.T. de Montpellier – Département informatique – 1ère année

Programmation 1 bis

TP : Simulation de circulation de tramways

1 Description

On se propose d'écrire un programme permettant de simuler des tramways circulant sur leurs lignes. La simulation porte sur plusieurs lignes et plusieurs rames (tramways) par ligne. Au cours de la simulation (où les secondes représentent des minutes dans la réalité), à chaque fois qu'une rame stoppe à un arrêt, la position du tram et son heure d'arrivée sont affichées.

Le programme est divisé en plusieurs classes :

- Une classe `Station` qui définit un objet station par son nom (de type `String`) et ses coordonnées x, y (`double`).
  Pour simplifier, les stations Corum des lignes 1 et 2 par exemple correspondent à des objets différents. On leur donnera des coordonnées légèrement différentes, simulant ainsi des rails disjoints.
- Une classe `Ligne` qui définit un objet par son `numero` (entier), sa `couleur` (`String`), un tableau `tabRames` qui permet de stocker l'ensemble de ses rames.
  Enfin, un attribut `tabStations` est un tableau des stations de la ligne ordonné selon l'ordre du parcours (le numéro d'une station sur la ligne est son indice dans le tableau).
- Une classe `Rame` qui définit l'objet rame par le numéro de station (`numStation`) où elle se trouve.
- Une classe `Simulation` qui définit un objet simulation par un tableau de `Ligne`. Son constructeur permet de créer « en dur » l'ensemble des lignes (et donc des rames) utiles à la simulation.
  Sa méthode `run` permet d'exécuter la simulation : toutes les minutes (secondes dans la simulation) à partir du temps $t = 0$, déplacement de toutes les rames de toutes les lignes de 1 station et affichage de l'ensemble des rames à la station où elles stoppent.
  L'affichage sera essentiellement basé sur l'exécution des méthodes `toString` des objets `Rame`.
- Une classe `TestSimulation` dont la procédure `main` définit un objet `Simulation` et lance sa méthode `run`.
  Par exemple, `java TestSimulation 150` lance la simulation pendant 150 pas de temps (minutes).

2 Premier programme : une seule ligne circulaire, une seule rame

Les lignes sont circulaires dans un premier temps et les rames tournent en boucle. Au temps 0, la première rame de la ligne quitte la station de départ (à l'indice 0 de `tabStations`) et tourne jusqu'au temps de fin de simulation.

Créer les classes ci-dessus et lancer une simulation.

Dans un premier temps, l'affichage se fera en mode texte, du genre :

```
temps 0 : ligne 1 station Corum
temps 1 : ligne 1 station Louis Blanc
...
```

Indication : simulation dans le temps

Utiliser la méthode de classe `Thread.sleep`. Par exemple,

```
try {
   Thread.sleep(1000); // timer de 1 seconde
} catch (InterruptedException e) {
   e.printStackTrace();
}
```

fait une pause de 1 seconde avant de passer à l'instruction suivante.

3 Plusieurs rames par ligne, puis plusieurs lignes

Prévoyez des sauvergardes avant chaque extension de votre programme courant.

Etendez votre programme pour accepter plusieurs rames par ligne, puis plusieurs lignes. Dans la simulation, le nombre de stations et le nombre de rames de la ligne permettent de faire partir les rames le plus régulièrement possible, par exemple :

(nb stations entre deux rames) = (nombre de stations) div (nbRames)

Il y a ensuite de nombreuses extensions possibles, réparties en deux grandes catégories :

- améliorations algorithmiques pour avoir une simulation plus réaliste ;
- conception d'une interface graphique pour améliorer la visualisation de la simulation.

4 Une simulation plus réaliste

Rentrer au garage

Une amélioration immédiate, mais pas si évidente, consiste à permettre aux rames de rentrer vers la station de départ après le temps de fin de simulation (pour la nuit par exemple). Pendant cette phase, les rames déjà présentes à la station de départ, ne doivent plus bouger, contrairement aux autres.

Suggestion : Ecrire une méthode calculant le nombre de pas de temps nécessaires pour que toutes les rames rejoignent la station de départ.

Des lignes aller-retour

Au lieu de gérer uniquement des lignes circulaires, vous pouvez étendre votre programme à des lignes aller-retour.

Suggestion : Ajouter un attribut `sens` à la classe `Rame` permettant de spécifier la direction courante de la rame. Le sens vaut +1 s'il s'agit d'une rame parcourant les stations de sa ligne dans l'ordre ou -1 si la rame parcourt les stations en sens inverse. Un attribut booléen `estCirculaire` de l'objet ligne spécifie son type (circulaire ou aller-retour).

5 Interface graphique

Ecrire une version améliorée de la méthode `run` qui permet un affichage graphique, un point s'affichant pendant une seconde aux coordonnées des arrêts effectués.

Vous trouverez sous l'ENT, dans le répertoire `graphisme`, un petit programme permettant d'afficher deux points qui clignotent reliés par un segment. Ce programme est formé de fichiers proposés par un de vos camarades, à partir de quelques lectures utiles sur le web, par exemple : `http://openclassrooms.com/courses/apprenez-a-programmer-en-java/notre-premiere-fenetre`.

Vous pouvez vous inspirer de ces classes, notamment `Point` et `Trait`, pour symboliser le trajet et les stations ; le mouvement d'un tramway peut être suggéré par le changement de couleur des stations.

6 Libre cours à votre imagination

Faites la version de vos rêves, simulez le réseau montpellierain par exemple, gérez les passagers qui montent et qui descendent, etc.