Simulateur

L'informatique peut être utile pour simuler des systèmes et aider à améliorer/optimiser ces systèmes. Pour ce projet, nous vous invitons à implémenter un petit simulateur de plusieurs ascenseurs. On gardera en tête l'exemple de dauphine. Que se passerait-il si le quatrième ascenseur était un jour réparé? Que se passerait-il si un ou plusieurs ascenseurs étaient reservés aux étages "haut" et ne s'arrêtaient pas aux étages 1, 2 et 3 à la montée? Pour avoir un début de réponse à ces questions, nous vous demandons donc d'implémenter un modèle simplifié des ascenseurs et de comparer quelques politiques d'utilisation différentes.

Ce projet est malgré tout un projet pour un cours de programmation objet en java. L'évaluation reposera donc en grande partie sur la qualité de votre architecture, l'utilisation des solutions offertes par java, la modularité de votre code (ex: il devrait être facile d'ajouter, d'enlever un ascenseur, il devrait être facile d'implémenter une autre politique d'utilisation des ascenseurs, l'architecture devrait rendre possible (et si possible simple) de modifier un jour le modèle pour le rendre plus réaliste, etc...). En second lieu sera apprécié les modèles plus réalistes.

On vous propose de discrétiser le temps, la simulation se fera un pas de temps à la fois. Durant un pas de temps, les ascenseurs pourront se déplacer, embarquer ou débarquer des personnes, ou rester à leur place. Pour simplifier, le temps pour qu'un ascenseur monte ou descende d'un étage sera proportionnel à une unité de temps. Le temps d'embarquement et de débarquement sera fonction du nombre de personnes qui embarquent, débarquent ou restent dans l'ascenseur. Pour manipuler cette fonction, on vous propose d'utiliser un objet qui implémentera l'interace suivante:

```
1  public interface TransferTime {
2        /**
3        * Method to compute the time needed for the people to exit the lift and enter the lift. This time depends on
         * the number of people leaving or entering the lift, but also the number of people that stay in the lift
         (the more people stay, the more time is needed to make the change).
4        * @param in is the number of people that enter the lift
5        * @param out is the number of people exiting the lift
6        * @param stay is the number of people that stay in the lift
         * @return return the waiting time
         **/
         public int compute(int in, int out, int stay);
   }
```

Pour une simulation, un certain nombre de personnes feront une requête pour aller d'un étage à un autre. Une personne sera donc identifiée par un identifiant, le pas de temps auquel la personne appellera l'ascenseur, l'étage où il se trouve et l'étage où il désire aller. La simulation s'arrête soit au bout d'un nombre de pas de temps maximum, soit quand toutes les personnes ont été transportées à l'étage souhaité.

Vous devrez finalement implémenter un système centralisé qui décide quel ascenseur doit se charger d'une requête. Ce système représente donc la politique avec laquelle vous utilisez vos ascenseurs. Attention, on essaiera ici d'être réaliste: la politique utilisera évidemment l'étage où se trouve la personne et si la personne souhaite monter ou descendre, mais évidemment, le système ne devrait pas connaître l'étage souhaité avant que la personne ne soit montée dans

un ascenseur. Par contre, pour prendre votre décision, vous connaissez la position de chaque ascenseur ainsi que leur direction.

Pour évaluer votre système, on vous propose de calculer le temps moyen d'attente de l'ascenseur. Vous pourrez proposer d'autres mesures si cela vous semble pertinent.

Le simulateur pourra fonctionner suivant deux modes:

- en lisant un fichier qui contient toutes les requêtes. Chaque ligne du fichier sera une requête de la forme:
  `id, step, startFloor, endFloor`
  où `id` est un **int** qui représente l'identifiant de la requête; `step` est **int** qui représente le pas de temps qui correspond à l'appel de l'ascenseur; `startFloor` et `endFloor` sont deux **int** qui correspondent à l'étage de départ et d'arrivée.

- en générant des requêtes aléatoires durant la simulation ou avant de lancer la simulation. On devra pouvoir paramétrer facilement les lois pour décider des arrivées des personnes, décider des étages de départ et d'arrivée. Par défaut, on pourra utiliser une loi de poisson pour modéliser l'arrivée des personnes, et choisir les étages de départ et d'arrivée de manière uniforme.

  Pour ce mode, on pourra donc réaliser k simulations de n pas de temps avec les mêmes paramètres de façon à calculer des moyennes et des écarts types qui aideront à mieux évaluer une politique.

Enfin pour ce projet, on vous demande de tester au moins deux politiques. Les deux politiques de base auront une base commune: c'est l'ascenseur le plus proche qui se déplace dans la bonne direction qui répond en premier à une nouvelle requête. La seconde politique aura une spécificité pour la montée: k ascenseurs seront dédiés au étages "haut" (s'il y a m étages, on va considérer qu'un étage haut est supérieur (strictement) à $\lfloor \frac{m}{2} \rfloor$. Pour la descente, on pourra considérés que les ascenseurs dédiées aux hauts étages peuvent s'arrêter à tous les étages, et qu'ils terminent toujours au rez-de-chaussée. Le problème du replacement des ascenseurs quand il n'y a pas de demande est laissé libre.

Vous êtes bien sûr invités à trouver de meilleures politiques! Ceci sera évalué pour des points bonus pour le projet.

Pour configurer la simulation, vous devrez lire un fichier texte nommé config.txt qui contiendra les informations suivantes (une infomation par ligne, dans l'ordre donné ci-dessous):

- nombre d'étages: les étages sont numérotés de 0 à n, on donnera la valeurs de n dans le fichier de configuration
- nombres d'ascenseurs: un entier strictement positif
- capacité des ascenceurs: un entier strictement positif
- vitesse des ascenseurs: un entier strictement positif qui indique le nombre de pas de temps pour passer d'un étage à l'étage directement suivant (i.e. passer de l'étage i à l'étage $i+1$ ou $i-1$).

Voici un exemple de fichier, il sera formatté de la sorte

```
nbFloors : 6
nbLifts : 4
capacity : 10
velocity : 5
```

On vous demande également de rédiger un petit rapport (2 pages maximum)

- Une justification pour la structure de votre code.
- Si vous proposez de meilleures politiques, expliquez votre politique et donner des éléments montrant ses qualités et défauts.
- Une discussion sur ce qui a été le plus dur à implémenter pour vous et sur le niveau de difficulté du projet.

Soumission

1. Le projet est à rendre le **5 janvier 2020** sur la plateforme `myCourse`. Tout retard sera pénalisé.
2. Vous devez soumettre deux fichiers:
   - une archive `zip` dont le nom est la concaténation des noms des auteurs et qui contiendra l'ensemble des fichiers sources. Si les auteurs sont Astérix et Obélix, le fichier zip sera donc nommé `asterix_obelix.zip`. Lorsqu'on décompresse le fichier zip, on devra trouver un répertoire nommé `asterix_obelix` contenant tous les fichiers sources. On devra pouvoir compiler et exécuter grace à la console votre code depuis l'intérieur de ce repertoire: si on se place dans le répertoire `asterix_obelix`, je dois pouvoir compiler et exécuter votre code. Si cette structure n'est pas respectée, vous serez pénalisé.
   - un document au format pdf contenant votre rapport nommé `asterix_obelix.pdf`. N'oubliez pas d'indiquez le nom des auteurs du travail dans le rapport.

   Vous serez pénalisé si vous ne suivez pas ces consignes.

Critères d'évaluation

Le projet est à effectuer de manière préférentielle en binôme. Si vous présentez un projet en trinôme, puisque votre force de travail augmente de 50%, la notation sera beaucoup plus sévère à travail égal.

L'évaluation portera sur la qualité de la structure de votre implémentation, sur la qualité de vos algorithmes (c'est un cours de programmation orientée objet), sur l'utilisation de ce qu'on a vu en cours, sur votre rapport, ainsi que sur une soutenance (qui sera organisée après la semaine d'examen). Cette soutenance ne demande aucune préparation de votre part. Elle durera une douzaine/quinzaine de minutes par groupe. La soutenance consistera en un échange au sujet de

vos résultats, votre rapport, et du code. Si la soutenance fait apparaître qu'un des membres n'a pas beaucoup contribué, sa note pourra être revue à la baisse.

Ce projet compte pour 40% de la note de l'UE. Il est donc souhaitable que la note corresponde au travail de votre groupe, et non aux conseils d'autres groupes, d'autres étudiants ou d'internet. Si vous utilisez des sources (articles de recherche, posts sur internet, etc...), vous devez mentionnez vos sources dans le rapport. Si j'ai des soupçons de plagiat, je n'hésiterai pas à saisir le conseil de discipline.