EFREI école d'ingénieur Paris Sud

Projet JAVA Niveau 2

Projet - Compagnie de taxis

Ce projet consiste à simuler graphiquement la gestion opérationnelle d'une compagnie de taxis. La compagnie dispose de taxis et de navettes. Lorsque la compagnie reçoit un appel, elle lui envoie un véhicule approprié. Les taxis vont chercher et transportent des passagers unitaires. Les navettes (véhicules effectuant de courts trajets répétitifs de point à point) sont des minibus ayant la capacité de transporter plusieurs personnes. Le territoire concerné est une ville représentée schématiquement et graphiquement par une grille rectangulaire.

Fichiers joints à cet énoncé

1) un répertoire *src* contenant
   a) les fichiers sources .java (en Java 1.7) constituant le noyau de code de départ (ensemble non immédiatement fonctionnel). La classe *Launcher* contient la classe de lancement *main*.
   b) le répertoire *images* des imagettes utilisées dans l'IHM
2) *projetCieTaxis_v0.pdf* : diagramme UML très succinct du jeu de classes fourni.
3) *projetCieTaxis_v0.jar* : exemple d'exécutable illustratif d'une version minimaliste. Pour une entrée en matière, commencer par lancer cet exécutable (puis patienter éventuellement un peu que des taxis se déplacent).

L'3 Projet JAVA Niveau 2 // 2016-2017

Travail à réaliser

Il s'agit de comprendre les classes java fournies puis de les compléter dans l'objectif de développer progressivement une simulation aussi riche que possible mais restant cohérente avec l'approche initiale. Le développement sera incrémental et enchaînera les phases spécifiées ci-après.
Le développement proposé utilise le *modèle MVC*, votre rapport devra en expliquer le rôle et le fonctionnement.

Phase 1

Comprendre le rôle de chaque classe et de chacun de ses membres (attributs et méthodes). Savoir lire et comprendre l'ensemble du code fourni (hormis la classe interne *CityView*).
Dans la classe *Location*, implémenter le corps de la méthode *nextLocation*.
Nota : à ce stade, le programme de simulation est censé être fonctionnel.
On souhaite attacher à chaque véhicule un identifiant de type *String* (par exemple pour un taxi : 'T' + un numéro spécifique à ce taxi). Compléter la classe *Vehicle* et la méthode appropriée de la classe *TaxiCompany* pour que chaque véhicule créé reçoive automatiquement un identifiant qui lui soit spécifique. Cet identifiant, une fois attribué, ne pourra plus être modifié.
Dans la classe *TaxiCompany*, modifier la méthode appropriée pour que ce soit le taxi libre le plus proche du demandeur qui lui soit envoyé.

Phase 2

Compléter la classe *CityGUI* pour afficher en sus, sur deux lignes et dans la même fenêtre graphique que celle déjà gérée, les informations suivantes : dimensions (largeur, hauteur) de la grille de la ville, nombre de taxis dans la ville, nombre de taxis libres, nombre de passagers en attente d'un taxi (toutes informations déductibles de *City*). Puis compléter cet affichage, sur une troisième ligne, avec les informations suivantes : valeur de *missedPickups* (définie dans la classe *PassengerSource*), valeur de l'attribut *idleCount* de chaque véhicule (défini dans la classe *Vehicle*).

Phase 3

Créer une classe *ConfigReader* permettant à l'utilisateur de saisir, dans une seule fenêtre graphique autonome, l'ensemble des paramètres de configuration de la simulation : nombre de taxis de la compagnie, nombre de navettes de la compagnie, dimensions (largeur, hauteur) de la grille de la ville, voire autres paramètres. La lecture sera totalement sécurisée.
Cette fonctionnalité sera utilisée par la classe *Simulation* pour caractériser les acteurs comme souhaité.
Dans la classe *CityGUI*, optimiser la taille de la fenêtre graphique initiale de façon qu'elle soit aussi grande que possible (quelle que soit la taille de l'écran) *mais* conserve un ratio hauteur / largeur égal à celui caractérisant la ville. Imposer aussi une taille minimale à la fenêtre graphique de façon que l'utilisateur ne puisse pas réduire exagérément la taille de fenêtre d'affichage.

Phase 4

Implémenter la classe *Shuttle* (navette) et compléter les autres classes pour permettre, en sus et sur le même panneau graphique que celui déjà géré, une simulation de navettes opérant sur la grille de la ville.

Phase 5

En sus des réalisations des phases précédentes, vous avez libre cours pour offrir des fonctionnalités complémentaires.
Par exemples :

1) gérer le cas où un client n'a pas de taxi immédiatement disponible mais attend d'en avoir un, gérer le cas où un client en attente finit par trouver le temps trop long et disparaît sans en informer qui que ce soit
2) incorporer une IHM clavier téléphonique permettant de simuler un appel téléphonique pour la réservation d'un taxi
3) afficher sous forme d'un histogramme *JFreeChart* le nombre de transports réalisés par chaque taxi
4) gérer plusieurs compagnies de taxis
5) définir et gérer une zone piétonne interdite aux véhicules
6) sauvegarder de façon automatique dans un fichier texte l'historique des simulations (ex : jour, heure, paramètres de la simulation, valeurs finales des variables d'état pertinentes), …

Contraintes de développement et d'implémentation

1) Le langage à utiliser est Java 1.7 minimum.
2) Tous les attributs des classes doivent être déclarés private (ou éventuellement protected si justifiable par des considérations d'héritage)
3) Aucun prototype de méthode ne doit être modifié (si nécessaire, ajouter une autre méthode de même nom (concept de surcharge)).
4) Sauf en phase de mise au point, aucune entrée/sortie n'est autorisée en mode console.
5) Ne pas utiliser de générateur d'interfaces graphiques.
6) Recoder l'interface graphique avec l'approche étudiée en cours (*thread* dans la méthode *main* avec *invokeLater/new runnable()* ..., méthodes *build* et *buildContentPane*, interface *ActionListener* ou classe *AbstractAction*).

Recommandations

1) Tester les fonctionnalités développées au fur et à mesure de leur développement!
2) A chaque phase du développement, garder une copie de la version opérationnelle à ce stade.

Équipes projet

Le projet est à réaliser en binôme. Les règles de constitution des binômes seront énoncées en début d'unité.

Critères d'évaluation du travail réalisé

Multicritères : opérationnalité, respect du cahier des charges et des contraintes, ergonomie, fonctionnalités complémentaires, pertinence des choix techniques, qualité de la programmation, capacité à répondre aux questions sur le code, contribution de chacun des membres du binôme au travail réalisé, qualité du rapport …

Evaluation en séance

Une partie de l'évaluation (démonstration, …) sera réalisée au cours de la dernière séance de TP-PRJ.