Programmation objet 2016-2017 : mini-projet "la Matrice"

Vous avez tiré la pilule rouge, bienvenue dans la réalité.
Vous êtes le technicien informatique chargé de créer un programme de gestion de vaisseaux et des équipages pour plonger dans **La Matrice**. La flotte de *Sion* vaincra si elle est capable de prendre le contrôle de **La Matrice**.
La flotte de *Sion* est en pleine déliquescence. Elle se limite aujourd'hui à quelques vaisseaux.
Chaque vaisseau est identifié de manière unique par un nom. En plus, il a un type (guerre, transport, etc.), et il contient un équipage de cinq membres au maximum.
Un membre d'équipage est une personne identifiée de façon unique par son nom ("Morpheus", "Neo", …). En plus, il possède les informations suivantes : sexe, grade (commandant, colonel, lieutenant, etc.).
On distingue deux types de personne: les "opérateurs *Sion*", des personnes chargées des vaisseaux (pilotage, entretien, maintenance, communication, . . .) et les "membres libérés", des personnes qui ont la capacité de pouvoir s'infiltrer dans **La Matrice**. Pour ce deuxième type, on comptabilise le nombre de fois qu'ils sont entrés et sortis de **La Matrice**. Les opérateurs ne peuvent pas s'infiltrer dans **La Matrice**.
Pour assurer la sécurité de l'équipage pendant une mission, il est important qu'au moins un opérateur *Sion* soit présent dans un vaisseau.

I) Mission 1

Nous vous demandons de définir un diagramme de classe UML pour représenter les données manipulées. Voici ci-dessous les fonctionnalités attendues :
- créer une personne (opérateur *Sion* ou membre libéré) puis l'ajouter dans la liste du personnel de *Sion*.
- afficher la liste du personnel.
- gérer les vaisseaux : créer un vaisseau puis l'ajouter dans la flotte de *Sion*.
- afficher l'ensemble des vaisseaux de la flotte (et des équipages associés).
- ajouter un membre du personnel dans un certain vaisseau.
- afficher l'ensemble des personnels affectés à un vaisseau.
- supprimer un membre d'un certain vaisseau. Il reste alors dans la liste du personnel de Sion.

Vous implémenterez en Java un programme de gestion de flotte (gestion des membres, des vaisseaux et des équipages). Le programme devra comporter un menu principal. Vous stockerez vos sources dans un répertoire *src* et vos *.class* dans un répertoire *bin*. Vous utiliserez des paquetages. Vous pouvez utiliser la classe *Liste* dans *ressource* sur *madoc*.

II) Mission 2

préambule : vous modifierez votre diagramme de classe en conséquence

Vous allez maintenant vous préoccuper des "infiltrations" dans **La Matrice**.
Seuls les membres d'équipage "libérés" figurant dans un vaisseau sécurisé (il a au moins un membre de l'équipage de type "opérateur *Sion*") peuvent s'infiltrer dans **La Matrice**.
Un membre infiltré sera placé dans une position (x,y) disponible (vide) où *x* et *y* sont deux entiers entre 0 et 9 calculés aléatoirement.
La Matrice comportera au moment de sa création 3 agents (des personnes). Ils seront automatiquement créés et placés. Ces agents ont pour propriétés :
- un nom ("agent_0" , "agent_1" , "agent_2")
- un sexe,
- un grade ("agent"),
- un degré d'efficacité (un entier entre 0 et 5) généré aléatoirement au départ,
- une position (x,y) dans la matrice calculée aléatoirement. x et y sont des entiers entre 0 et 9.

Le programme devra, dans cette mission, permettre de gérer **La Matrice** à travers les fonctionnalités suivantes :
- afficher les membres d 'équipage pouvant s'infiltrer.
- infiltrer un membre dans **La Matrice**.
- sortir un membre de **La Matrice**.
- afficher la liste des membres dans **La Matrice** avec leurs positions (x,y) respectives.
- afficher si la flotte de Sion a réussi ou non à vaincre.
- afficher **La Matrice**.

Voici ci-dessous un exemple d'affichage de **La Matrice** sur le terminal. On affichera : " " si la position est vide, "A" suivi d'un entier s'il y a un agent, l'entier représente son degré d'efficacité. "M" ou "m" s'il y a un membre infiltré.

	0	1	2	3	4	5	6	7	8	9
0										
1	A2									
2										
3								A0		
4				M						
5								M		
6										
7					A3					
8										
9										

La vie dans la Matrice

Ecrire une méthode *distanceAgent(...)* pour calculer la distance (un réel) entre un membre *m* et un agent *a* et une méthode *agentPlusProche(...)* pour déterminer l'agent le plus proche d'un membre *m* dans **La Matrice**.

Quand un membre est infiltré soit il est "infecté", soit l'efficacité de l'agent le plus proche est diminuée.

Pour cela, écrire la méthode *estInfecte(...)* permettant de vérifier si *m* est "infecté". Son pseudo-code est le suivant :

soit *a* l'agent le plus proche d'un membre *m*
si (("efficacité de *a*" / "distance entre *a* et *m*") > "nbEntreesSortiesMatrice de *m*")
alors *m* est infecté
sinon *m* n'est pas infecté

- Si le membre est infecté lors de son infiltration, alors il sera "déconnecté", c'est à dire qu'il ne pourra plus sortir de **La Matrice**. A l'affichage de **La Matrice,** "m" (minuscule) apparaîtra à sa position.
- S'il n'est pas infecté, alors l'efficacité de l'agent le plus proche sera diminuée de moitié. Il pourra sortir de **La Matrice**, son nombre d'entrées et sorties sera incrémenté lors de sa sortie et à l'affichage de **La Matrice** "M" apparaîtra à sa position.
- La flotte de *Sion* vaincra si elle prend le contrôle de **La Matrice**. C'est à dire si tous ses agents sont neutralisés (leur degré d'efficacité est zéro). Ce résultat sera signalé lors de l'affichage de **La Matrice**.

III) Mission 3 (bonus)

Donner libre cours à votre imagination pour introduire d'autres éléments dans **La Matrice**.

A rendre

- vous travaillerez en binôme.
- vous stockerez votre code dans un répertoire nommé *nom1_nom2* qui contiendra un répertoire *src* où seront stockés vos sources et un répertoire *bin* où seront stockés vos .class.
- il faudra utiliser des paquetages pour structurer votre programme.
- vous devez rendre, en plus des sources (commentés => javadoc) de votre programme, un compte-rendu en *pdf* analysant vos choix d'implémentations. Il contiendra:
  - le diagramme de classes UML
  - une description des fonctionnalités implantées, difficultés rencontrées, etc
  - un jeu d'essai, avec des copies d'écran, montrant le fonctionnement de votre application
  - un descriptif donnant les commandes pour compiler et exécuter votre application.

Vous devez rendre sous *madoc*, une archive nommée *nom1_nom2.tar.gz* au plus tard **samedi 8 avril 2017 à minuit.**

nb: Ce mini-projet est largement inspiré de http://mmi.univ-smb.fr/~mtoma/info323/TPs/Tp2_3/