DS Ei3-INFO - OBJET

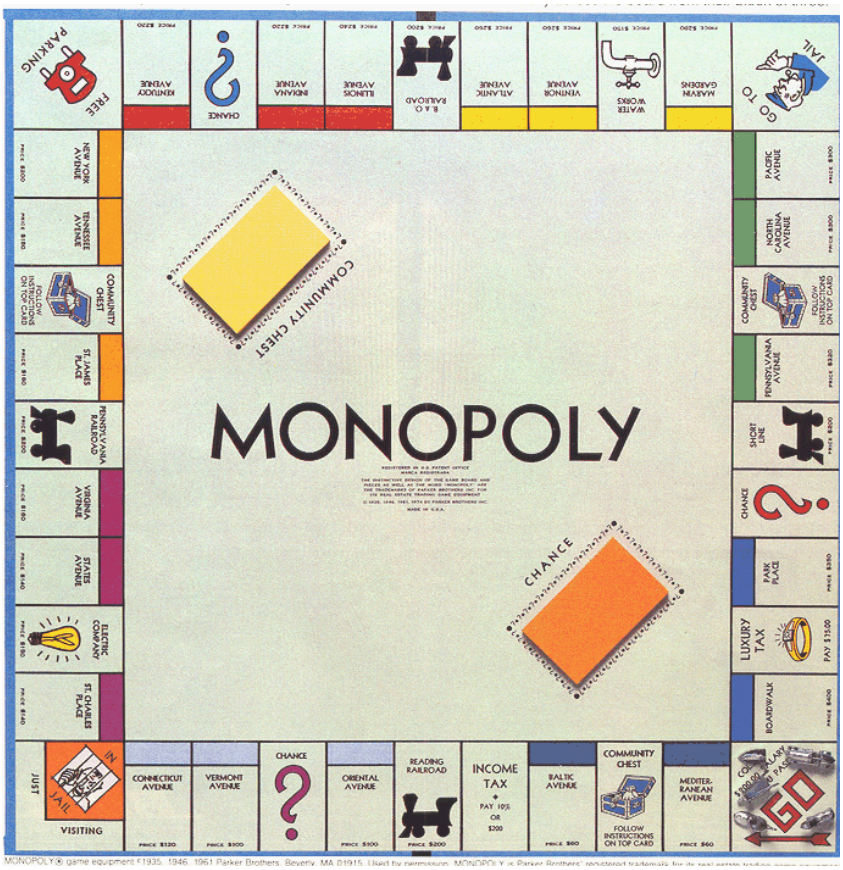

Figure 1: plateau de Monopoly standard

Le but de ce DS est de recréer un jeu de Monopoly (simplifié) en parcourant l'ensemble des concepts objet vus pendant le cours. Il est fortement recommandé de lire l'ensemble de l'énoncé avant de se jeter sur la première question.

1.1. Création des cases

Le plateau de jeu se compose de 40 cases qui ont des rôles différents. On trouve deux grandes catégories de cases : celles qui sont achetables (gares, emplacements constructibles) et celles qui ne le sont pas (prison, départ, cases chance…).

1.1.1. Diagramme des classes *Case*

Ecrire un diagramme de classes permettant la mise en place des catégories de cases suivantes qui dérivent d'une classe abstraite *Case* (toutes les cases sont nommées) :

- des cases *Achetables* celles-ci auront un prix et un propriétaire qui pourra être modifié via l'opération *acheter()* ; le propriétaire sera représenté par une classe *Joueur* que nous définirons plus tard.
- Des cases spécifiques pour les *Gares* ;
- Des cases spécifiques pour les lieux *Constructibles* (sur lesquelles on peut mettre des maisons voire des hôtels). Prévoir de quoi calculer le loyer en fonction du nombre de maisons ou d'hôtels.

Remarques :

1. par rapport au jeu habituel, on ne créée pas de couleurs de case pour regrouper les lieux (exemple : Belleville, Lecourbe dans le plateau classique).
2. La plateau de jeu sera affiché en appelant la méthode toString() de chacune des cases. Elle doit figurer là où c'est approprié dans chacune de cases.
3. Prévoir des cases larges pour compléter le diagramme au fur et à mesure du DS

1.1.2. Méthode(s) toString()

Ecrire le code java de méthodes toString() définies dans la question précédentes à l'aide des exemple suivants :

```
Gare de Lyon (coût : 20000 €)  - sans propriétaire

Départ

Rue Crébillon (coût : 45000€) – propriétaire : Bidule, 2 maisons, loyer =
1500€
```

1.1.3. Calcul des loyers (simplifié)

Le calcul du loyer (coût du passage sur la case) s'effectue de la façon suivante :

- si la case n'a pas de propriétaire, le loyer est nul ;
- si la case a un propriétaire qui n'est pas le joueur qui tombe sur cette case, le loyer est
    - fonction du nombre de maisons et d'hôtels pour les cases constructibles (on utilisera deux coefficients *a* et *b* supposés connus)
    - 2500 fois le nombre de gares possédées par le propriétaire de la case en question[1]

1.2. Plateau de jeu

Le plateau de jeu contient 40 cases stockée sous la forme d'une *ArrayList<Case>* nommée *plateau*. On considère qu'il contient une méthode *initPlateau()* qui initialise le tableau. Il contient aussi un attribut *joueurs* qui est une *LinkedList<Joueur>*.

1.2.1. Calcul du nombre de gares

Ecrire une méthode *nbGares()* qui calcule le nombre de gares possédée par un joueur passé en paramètre.

1.2.2. Affichage du plateau

Ecrire une méthode Affiche() qui affiche le plateau de jeu en utilisant uniquement les méthodes *toString*() des cases.

1.2.3. Avancer sur le plateau

Ecrire une méthode *avance()* qui prend en argument une case et qui avance de *d* cases, c'est-à-dire qui renvoie la case située *d* cases plus loin.

---

[1] Utiliser la méthode *nbGares()* qui sera définie en 1.2.1.

1.3. Les joueurs

1.3.1. Définition

Ecrire une classe Joueur qui comprend les attributs nécessaires pour représenter son nom, sa fortune ainsi que sa position sur le plateau et une référence au plateau. On prévoira également un constructeur qui initialise convenablement ces données qui seront passées par paramètre (sauf la fortune qui sera initialisée à 100000€). Le nombre de gares devra également pouvoir être calculé depuis la classe Joueur.

1.3.2. Paiement d'une somme à un autre joueur

Ecrire une méthode permettant le paiement d'une somme *x* à un joueur *j*. En cas d'insuffisance de crédit, la méthode devra lancer une exception *NoMoreMoney* qui est également à écrire.

1.3.3. Tour de jeu

Ecrire une méthode *tourDeJeu()* qui effectue les opérations suivantes :

1. Lance le dé à l'aide de la méthode statique définie ci-dessous :

```
public static int lanceLeDe() {
     return ((int) Math.floor(Math.random()*6))+1;
}
```

2. calcule la nouvelle case courante, affiche le message « Le joueur XX est en YY »
3. si la case est de type *Achetable* l'achète si le coup effectué précédemment par le dé est impair (et si elle n'a pas encore de propriétaire et que le joueur en a les moyens !)
4. si la case est de type *Achetable*, paie le loyer dû au propriétaire

Précision : cette méthode ne se préoccupe pas des éventuels problèmes de paiement.

1.4. Retour sur le plateau

1.4.1. Fin de la partie

Ecrire une méthode *findePartie()* qui renvoie un booléen vrai lorsque la partie est terminée.

1.4.2. Modification de la méthode *affiche()*

Ecrire le code à ajouter dans la méthode affiche() de la question 1.2.2 pour afficher la liste des joueurs, l'état de leur finances ainsi que la liste des cases en leur possession.

1.4.3. Tour de jeu complet

Ecrire une méthode *tourDeJeu()* qui fait jouer l'ensemble des joueurs tant que la fin de la partie n'est pas détectée. Les joueurs qui ont des défauts de paiement sont supprimés de la liste *joueurs*. Un message s'affiche à chaque élimination de joueur.