TD d'algorithmique M2202 – TD7 à 9 – Programmation objet

Rappels de cours :

Le type `String` vu précédemment est en fait une **classe** prédéfinie du langage Java qui permet de manipuler des suites de caractères (des « chaînes de caractères »).

La classe `String` fournit des méthodes qui permettent la recherche de mots, ou de caractères, bref, de sous-chaînes, dans un texte (méthode `substring`). Les mots peuvent aussi être comparés suivant l'ordre alphabétique (méthode `compareTo`, qui renvoie 0 si la chaîne de l'objet est égale à celle fournie en entrée, une valeur positive si la chaîne de l'objet est supérieure à celle fournie en entrée, une valeur négative sinon). La liste complète des méthodes de la classe String est décrite sur la page http://docs.oracle.com/javase/7/docs/api/java/lang/String.html. Voici par exemple la documentation fournie pour les méthodes `substring`, `compareTo` et `compareToIgnoreCase` :

`int` `length()` Returns the length of this string.	`String` `substring(int beginIndex)` Returns a new string that is a substring of this string.
`int` `compareTo(String anotherString)` Compares two strings lexicographically.	`String` `substring(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.
`int` `compareToIgnoreCase(String str)` Compares two strings lexicographically, ignoring case differences.	

L'étude des objets de type `String` nous montre qu'une classe est une association de données appelées « **propriétés** » (information, valeur de tout type) et de fonctions appelées « **méthodes** » (outils d'accès et de transformation des données). Ces méthodes, définies dans une classe, ont directement accès aux données de cette même classe.

Le langage Java offre aussi la possibilité à la personne qui programme de développer ses propres classes. Construire une classe, c'est définir un nouveau type. Pour cela, il est nécessaire de :
• déterminer les caractéristiques communes à ce que l'on souhaite décrire. Ce sont les **propriétés**, c'est-à-dire les données, les attributs, ou encore les membres de la classe.
• définir les **méthodes**, c'est-à-dire toutes les opérations et traitements, réalisables sur ces éléments. Ces opérations sont aussi appelées comportements.

Une classe définissant un type structuré n'est pas une application directement exécutable. Elle ne contient pas de fonction `main`.

Les types structurés sont utilisés dans les applications, en déclarant des variables dont le type correspond au nom de la classe définie précédemment, comme le montre l'instruction suivante :

```
TypeDeLObjet chose = new TypeDeLObjet();
```

L'opérateur **`new`** détermine l'adresse où stocker les informations relatives à l'objet déclaré. Il réserve l'espace mémoire nécessaire pour stocker les données et les méthodes de la classe. Les données sont initialisées à 0 pour les entiers, 0.0 pour les réels, '\0' pour les caractères et `null` pour tous les autres types structurés.

À cette étape, la variable `chose` contient ce qu'on appelle dans le jargon informatique un **objet**. Un objet est donc un élément particulier, un représentant d'une classe définissant un type structuré. On dit aussi que c'est une **instance** de la classe. Les données (propriétés ou attributs) qui définissent cette instance sont stockées dans les variables de classe associées à cette instance particulière, appelée **variables d'instance**.

L'accès aux variables d'instance ainsi qu'aux méthodes de la classe se fait par l'intermédiaire de l'opérateur « . » (le point). On peut le voir dans l'exemple suivant, en supposant que la donnée `nomDeLaDonnee` et la méthode `nomDeLaMethode` ont été préalablement définies dans la classe `TypeDeLObjet` de l'objet `chose` :

```
// affectation d'une valeur dans la propriété de l'objet chose :
chose.nomDeLaDonnee = [valeur du bon type] ;
// appel de la méthode sur l'objet chose :
chose.nomDeLaMethode([liste des paramètres éventuels, précédés de leur type]) ;
```

Exercice : le jeu du Monopoly

On souhaite lors de ces séances construire un jeu de Monopoly simplifié. Pour cela, on va créer des classes `Joueur`, `CaseMonopoly` et `Monopoly`. Vous ajouterez progressivement le contenu dans chaque classe en fonction des questions.

Partie 1 – Les cases du jeu

Q1. Créez une classe `CaseMonopoly` en précisant quel sera le nom du fichier (laissez 2 lignes vides avant le début).

Q2. Ajoutez des propriétés à cette classe : le numéro de case, le nom, le type (terrain, gare, compagnie ou autre), la couleur, le prix, le nombre de maisons construites (on considère qu'un hôtel est équivalent à 5 maisons).

Q3. Ajoutez un constructeur à cette classe, qui reçoit en entrée les valeurs attendues pour toutes les propriétés de la classe.

Q4. Ajoutez une méthode `toString` qui renvoie une chaîne de caractères contenant le nom de la case, suivi, entre parenthèses, de son prix si on peut l'acheter, et s'il s'agit d'un terrain, de sa couleur et du nombre de maisons construites.

Q5. Ajoutez une méthode `coutMaison` à la classe `CaseMonopoly`, qui renvoie le coût d'une maison en fonction de la couleur de la case : 5 000 si rose ou bleu clair, 10 000 si violet ou orange, 15 000 si rouge ou jaune, 20 000 si vert ou bleu foncé.

Partie 2 – Les joueurs

Q1. Créez une classe `Joueur` en précisant quel sera le nom du fichier (laissez 2 lignes vides avant le début).

Q2. Ajoutez des propriétés à cette classe : le nom du joueur, sa position, son argent, un tableau de 40 cases contenant ses possessions (chaque case contient un objet de la classe `CaseMonopoly` s'il la possède, sinon elle contient `null`) et un tableau d'entiers `numerosPossessions` contenant les numéros de case de ses possessions.

Q3. Ajoutez un constructeur à cette classe, qui reçoit en entrée les valeurs attendues pour toutes les propriétés de la classe (initialement, chaque joueur est à la case 0, a 150 000 euros et ne possède rien).

Q4. Ajoutez une méthode `toString` qui renvoie une chaîne de caractères contenant le nom du joueur suivi, entre parenthèses, de son argent.

Q5. Ajoutez une méthode `aPerdu` qui renvoie FAUX si l'argent du joueur est supérieur à 0 euro, VRAI sinon.

Q6. En supposant que vous disposez d'une méthode `nombreALeatoire` qui prend en entrée un entier `min` et un entier `max` et qui renvoie un nombre entier aléatoire entre `min` et `max`, ajoutez une méthode `lanceDes` qui renvoie un tableau de deux cases, dont chaque case contient un nombre entier aléatoire entre 1 et 6 après avoir affiché ces deux nombres.

Q7. Ajoutez une méthode `acheteMaison` qui affiche au joueur la liste de ses possessions qui correspondent à des terrains et qui ont moins de 5 maisons (avec pour chacune le nombre de maisons construites et le coût d'une maison) et lui demande d'entrer un numéro de case s'il veut construire une maison sur cette case. Tant que le joueur entre un numéro de case qui correspond à un terrain, qui lui appartient et où il y a moins de 5 maisons, l'achat de la maison est effectué.

Partie 3 – Début de partie et tours de jeu

Q1. Créez dans la classe `Monopoly` une fonction `main` qui demande à l'utilisateur le nombre de joueurs et joueuses, puis leur nom, crée l'objet correspondant à chaque joueur, et stocke tous ces objets dans un tableau `joueurs`.

Q2. Créez dans la fonction `main` une variable `joueurActuel`, initialisée à 0, qui stockera le numéro de case du tableau correspondant à la personne en train de jouer, une variable `nbTours` initialisée à 0 qui stockera le nombre total de tours du jeu et une variable `aFaitUnDouble`, initialisée à `false`, qui indiquera si la dernière personne qui a lancé les dés a fait un double (et doit donc rejouer).

Q3. Créez dans la fonction `main` une boucle telle que chaque passage dans la boucle correspond à un tour de jeu. La boucle s'arrêtera donc quand le tableau `joueurs` n'a plus qu'une seule case. Au début de cette boucle, ajoutez le code nécessaire pour que la personne qui doit jouer lance les dés, et pour déplacer son pion sur le tableau du jeu.

Q4. À la fin de la boucle de la fonction `main`, ajoutez le code nécessaire pour mettre à jour (éventuellement) la variable `joueurActuel`.

Q5. Dans la boucle de la fonction `main`, ajoutez le code permettant de demander à la dernière personne qui a lancé les dés si elle souhaite acheter des maisons sur ses terrains.