POKÉTUDIANTS

Le but de ce TP est de modéliser la gestion d'une promotion de Pokétudiants. Un pokétudiant est défini formellement de la manière suivante :

Définition 1 (Pokétudiant) *Les pokétudiants sont des petits monstres possédant des aptitudes magiques.*

Un enseignant-dresseur peut garder avec lui un maximum de 4 pokétudiants, les autres devant rester dans le poké-bde. Un pokétudiant possède un nom ainsi qu'un maximum de trois aptitudes parmi les suivantes : TRAVAILLER, REVISER, DORMIR, BOIRE_CAFE, BAVARDER, SECHER, PAYER_PROF, EXPLOSER. À chaque aptitude est associée une utilité de la manière suivante :

aptitude	utilité	aptitude	utilité
PAYER_PROF	100	BOIRE_CAFE	-1
TRAVAILLER	50	DORMIR	-10
REVISER	20	BAVARDER	-50
SECHER	0	EXPLOSER	-500

Table 1: Aptitudes avec l'utilité associée.

Afin de modéliser les différentes aptitudes ainsi que les utilités de ces dernière, nous allons utiliser un type énuméré (avec le mot clef enum). Un *enum* simple s'écrit de la manière suivante en Java :

```
public enum Jour{
  LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE;
}
```

Pour utiliser un type énuméré il suffit de déclarer un attribut du type déclaré et de l'initialiser avec une des valeurs listées. Nous avons donc pour notre exemple de jour de la semaine :

```
public enum Test{
  public static void main(String[] args) {
    Jour j = Jour.LUNDI;
  }
}
```

Il est aussi possible d'associer des attributs à chaque élément de type énuméré que nous avons créé. Pour cela il suffit d'inclure les attributs et les méthodes que l'on souhaite associés aux éléments de notre type énuméré. Cela est réalisé de la manière suivante :

```
public enum Planete{
  MERCURE(3.303e+23),
  TERRE(5.976e+24);
  private final double masse;
  Planete(double masse){
    this.masse = masse;
  }
  public double getMasse(){return masse;}
}
```

1. Définissez une énumération afin de modéliser les aptitudes des pokétudiant.
2. Définissez une classe `PokEtudiant` (qui regroupe donc un nom, et un maximum de trois aptitudes).

Les pokétudiants sont généralement regroupés dans des structures afin d'éviter la solitude. Ces structures sont de deux types : le poké-bde et le poké-bed (bed pour bureau enseignant-dresseur). Quelque

soit la structure considérée il faut pouvoir ajouter un pokétudiant ou supprimer un pokétudiant. Afin de factoriser les similitudes des différentes structures pouvant accueillir les pokétudiants nous allons utiliser la notion de classe abstraite, en utilisant le mot clef `abstract`. Une classe abstraite est quasiment une classe normale. Cela dit, elle a tout de même une particularité : vous ne pouvez pas l'instancier ! Par exemple, considérons la classe `Animal` :

```
abstract class Animal{
  abstract void manger(); // une méthode abstraite
}
```

Imaginez que vous êtes en train de réaliser un programme qui gère différents types d'animaux. Vous aurez des loups, des chiens, des chats, des lions et des tigres. Nous n'allons pas faire recopier bêtement le code des différents animaux. En effet, tous ces animaux ont des points communs ! Et qui dit points communs dit héritage.

```
public class Chien extend{
  public void manger(){/* du code pour manger */ }
}
```

3. Écrivez une classe abstraite `PokeZone` qui gère de manière générale les structures dans lesquelles sont parqués les pokétudiants. Cette classe possède des méthodes abstraites pour l'ajout et l'enlèvement d'un pokétudiant d'une zone.

4. Le poké-bde est représenté par une classe contenant un tableau de pokétudiant et un entier stockant le nombre de pokétudiant dans le poké-bde. Cette classe étend la classe `PokeZone` et implémente donc les méthodes abstraites (pour le moment le corp de la méthode est vide). Sa capacité initiale est de 4 éléments. Écrivez un constructeur qui retourne un poké-bde vide.

5. Écrivez une méthode dans la classe qui gère le poké-bde qui prend en paramètres un pokétudiant, et qui place ce dernier dans la zone (Agrandissez la taille de le poké-bde si nécessaire, la capacité de celle-ci devant être un multiple de 4 . . . c'est mieux pour la belote).

6. Les pokétudiants qui restent avec l'enseignant-dresseur sont stockés dans un tableau. Écrivez une classe pour modéliser le bureau de l'enseignant-dresseur. Ajouter une méthode qui prend en paramètres le poké-bde et un pokétudiants, et qui ajoute ce dernier :

   - dans l'équipe si elle n'est pas remplie, ou si le pokétudiant a une utilité supérieure au pokétudiant de l'équipe ayant la plus faible utilité ; dans ce cas, ce dernier est reversé dans la poké-bde.
   - dans le poké-bde sinon.

   En ce qui concerne la méthode d'enlèvement d'un pokétudiant nous laisserons le corps de cette dernière vide . . . pour le moment.

6. Il arrive qu'un pokétudiant disparaisse sans laisser de trace. Par conséquent, il est nécessaire de le supprimer de la zone dans laquelle il se trouve. Afin de faciliter la suppression d'un pokétudiant de la zone dans laquelle il se trouve il est nécessaire d'ajouter un attribut au pokétudiant permettant de connaître l'endroit où il se trouve et d'implémenter la suppression de ce dernier dans la classe qui gère la zone. À la fin de la suppression :

   - si il reste au maximum 4 pokétudiants, ils se trouvent tous dans l'équipe de l'enseignant-dresseur.
   - si il reste plus de 4 pokétudiants, les quatre les plus utiles doivent être dans l'équipe, les autres dans la poké-cafet.
   - dans tous les cas, la capacité de la poké-cafet doit être le multiple de 4 le plus petit possible qui soit supérieur ou égal au remplissage de la poké-cafet ; la capacité ne doit cependant jamais être inférieure à 4.