Licence Informatique 2ème année

Programmation Objet I

TD n°3 - *Abstraction*

1- Au commencement était l'objet ...

Le 4e jour, Dieu commença la création des animaux en définissant les classes suivantes :

```
class EtreVivant {
        String nom;
        EtreVivant(String nom) {
                this.nom = nom;
        }
}


class Animal extends EtreVivant {
        Aliment[] bouffe;
        Animal(String nom, Aliment[] t) {
                super(nom);
                this.bouffe = t;
        }
        void manger() { ... }
}
```

1.1 Au matin du 5e jour, Dieu, débordé, confia une partie de la Création à l'ange Génilogiciel. L'ange devait définir les classes Mammifère, AnimalVolant et AnimalMarin. Les mammifères ont pour particularité une couleur de poil et une température corporelle. Les animaux volants ont tous une méthode pour voler et les animaux marins ont tous une méthode pour nager. Il doit être possible de créer des mammifères marins, comme les baleines, et des mammifères volants, comme les chauve-souris. Comment l'ange Génilogiciel s'y est-il pris ?

1.2 À la fin de la matinée du 5e jour, Dieu vint voir où en est Génilogiciel, et décida qu'en plus les animaux marins seraient caractérisés par un nombre de nageoires et les animaux volants par un nombre d'ailes. Comment l'ange a t-il fait pour satisfaire le Créateur ?

1.3 L'après-midi du 5e jour, Dieu, d'humeur farceuse après un bon déjeuner, décide d'ajouter à la Création une créature improbable, le poisson-volant. Comment l'ange Génilogiciel a t-il ajouté cette créature ?

2- Contes en tiroir

On veut écrire un programme pour représenter les Contes des 1001 Nuits. Un conte est composé d'un début (chaine de caractères), d'une fin (chaine aussi), et d'un certain nombre de parties stockées dans l'ordre. Chaque partie peut être une chaine de caractères, mais aussi une histoire en elle-même : les 1001 Nuits racontent l'histoire de Shéhérazade, qui elle-même raconte des histoires dans lesquelles des personnages racontent des histoires, dans lesquelles des

personnages racontent des histoires, etc. Tous les éléments des contes (histoires, parties, débuts et fins) doivent implémenter l'interface suivante :


```
public interface Racontable{
        public void print(); // écrit l'objet à l'écran
}
```

2.1 Proposez du code objet pour représenter les Contes des 1001 Nuits, sachant que la classe String est final (aucune classe ne peut en hériter).

2.2 Écrivez une méthode principale qui crée une histoire commençant par "*il était une fois*", se terminant par "*fin*", ayant pour première partie "..." et pour deuxième partie une histoire qui commence par "*ce jour-là*", se termine par "*à jamais*" et a pour unique partie "...".

3- Système de fichiers

On veut écrire un programme pour représenter le fonctionnement d'un système de fichiers. Un programmeur a déjà écrit l'interface suivante pour manipuler les éléments du système :

```
interface SystemElement{
        abstract boolean open(); //renvoie vrai si l'ouverture se passe bien
}
```

Il est *nécessaire* que les classes qui représentent les éléments du système implémentent cette interface. Les éléments du système peuvent être de types variés : répertoire, fichier texte, fichier exécutable, lien symbolique, ... Tout élément du système a un nom, une taille (en nombre d'octets) et un répertoire qui le contient.

Un répertoire a aussi une liste des éléments (fichiers, répertoires, ...) qu'il contient. La taille d'un répertoire est la somme des taille des éléments qu'il contient. Un répertoire possède une méthode *boolean mkdir(String n)* qui crée un sous-répertoire de nom n, une méthode *boolean touch(String n)* qui crée un fichier texte vide (de taille 0) dans le répertoire et une méthode *boolean addFile(File f)* qui ajoute un fichier donné dans le répertoire. Ces méthodes renvoient vrai si l'action se déroule correctement, faux sinon. Ici, on néglige les problèmes de nommage, c'est-à-dire qu'on peut autoriser l'existence d'éléments de même nom dans un même répertoire. Les méthodes *mkdir*, *touch* et *addFile* renvoient donc toujours vrai. Ouvrir un répertoire (méthode *open*()) consiste à afficher à l'écran la liste des noms et taille des éléments contenus dans le répertoire.

3.1 Écrire du code objet pour représenter les répertoires.

Un fichier a une taille propre (qui n'est pas calculée à partir d'autres tailles), et possède une extension qui dépend de son type. Les fichiers texte ont tous l'extension ".txt", les fichiers java une extension ".java", etc. Quand on crée un fichier, si le nom fourni ne finit pas par l'extension prévue, le constructeur doit ajouter l'extension au nom du fichier. Pour tester si une chaine de caractère finit par une certaine autre chaine, on peut utiliser la méthode boolean endsWith(String s) de la classe String. Ouvrir un fichier consiste à appeler un programme sur le fichier, à l'aide d'un objet de type Runtime.

Exemple : *java.lang.Runtime.getRuntime().exec("gedit toto.txt").exitValue()* va ouvrir le fichier toto.txt dans l'éditeur Gedit et retourner un code d'exécution qui vaut 0 si tout se passe bien.

3.2 Écrire du code pour représenter les fichiers en général et les fichiers texte en particulier.

4- <u>Membre de classe</u>

On veut compiler les classes Essai et Test. Que va afficher le compilateur?

```
class Test {
        int i;
        static int cpt = 0;
        static Test() {
                this.i = 1;
                this.cpt++;
        }
        static void plusUn() { this.i++; }
}

class Essai {
        public static void main(String[] toto) {
                Test t = new Test();
                plusUn();
                System.out.println(t.i);
        }
}
```

5- <u>Immatriculation</u>

On veut écrire une classe qui représente les immatriculations de véhicule. La première immatriculation est AA-1-AA. L'immatriculation suivante est obtenue en incrémentant en priorité le numéro, puis les lettres de droite et enfin les lettres de gauche. Quand le numéro atteint 999, sur la plaque suivante il repasse à 1 et on change les lettres de droite (en passant à AB, puis AC, ..., AZ, BA, ...).

5.1 Écrire une classe qui permet de représenter les immatriculations et de générer automatiquement les immatriculations, dans l'ordre, à chaque appel du constructeur. On peut supposer qu'il existe une méthode *static String nextLetters(String s)* qui renvoie les lettres qui suivent s (cette méthode renvoie "B" si s vaut "A", "AA" si s vaut "Z", "BA" si s vaut "AZ", etc).

5.2 On veut maintenant réutiliser les immatriculations des véhicules qui sont envoyés à la casse. Modifier la classe précédente pour rendre cette réutilisation possible.