TP2 Java : Programmation Orientée Objet

Instructions pour l'envoi de votre code

1. Vous êtes libres de discuter entre vous mais le code est *individuel*.
2. Chaque méthode (fonction) doit être testée sur au moins deux exemples. Inclure les tests dans votre archive.
3. Faire une archive (`.zip` ou `.tar`) de tous vos fichiers sources (`.java`) et l'envoyer à `omar.fawzi@ens-lyon.fr` avant samedi 27 septembre 23h59. Merci d'inclure dans le sujet `[Proj1] [TP2]` et de nommer votre archive `NomPrenomTP2.zip`

Ressources

Vous pourrez utiliser les resources suivantes :
– http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-java/votre-premiere-classe
– http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

1 Classes et héritage

Un objet est l'association d'un état et d'un comportement. L'état est représenté par des **champs** (attributs ou variables) et le comportement est exposé via des **méthodes** (ou fonctions). Une classe est une description abstraite d'un ensemble d'objets "de même nature". C'est un modèle pour la création de nouveaux objets. Désormais nous allons créer une classe par fichier (qui va porter le même nom).

Exercice 1 *Créer une classe* `Auteur` *avec les champs suivants :* `String nom`, `String prenom`, `String email`, `int age` *et une méthode* `String toString()` *qui renvoie une chaine de caractères contenant toutes les informations sur l'auteur.*
Créer une classe `Livre` *avec les champs suivant* `String titre`, `Auteur auteur`, `double prix`, `int nbEnStock`, `int nbVendus` *et* `int note` *et une méthode* `String toString()` *qui renvoie le titre et l'auteur du livre.*
Notez qu'aucune des ces deux classes ne contient une méthode `static void main(String args[])`. *Et c'est normal car dans une application il n'y a qu'un seul programme principal. Pour tester ces classes, créer une classe TestLivre qui teste ces deux classes. Pour construire un objet de type* `Livre`, *on écrit* `Livre l = new Livre()`.

La fonction `Livre()` est appelée constructeur. On peut *surcharger* une fonction (i.e., définir une fonction avec le même nom mais différents paramètres) en rajoutant des paramètres. Le constructeur a toujours le même nom que la classe.

Exercice 2 *Créer deux autres constructeurs pour chacune des classe* `Livre` *et* `Auteur`.

Les méthodes manipulent l'état interne de l'objet et servent à la communication inter-objets : c'est le principe d'*encapsulation*. On protège l'accès à l'état interne de la classe en rajoutant le mot clé `private` à un attribut ou une méthode. On crée ensuite des fonctions appelées "getters" et "setters" pour interagir avec les attributs de l'objet.

Exercice 3 *Modifier le code pour les classes de l'exercice précédant pour "encapsuler" les attributs internes de la classe. A part les getters et setters pour chaque attribut, on pourra créer une fonction* `boolean enStock(int n)` *qui renvoie* `true` *si il y a au moins n copies en stock et une fonction* `void vendre(int n)` *qui fait décroître l'attribut* `nbEnStock` *d'une valeur de n et augmenter l'attribut* `nbVendus` *d'une valeur de n.*

L'héritage est également un principle important. On utilise le mot clé `extends`.

Exercice 4 *Créer une classe* `Dictionnaire` *qui hérite de* `Livre` *et a deux attributs en plus* `String langue` *et* `int nbMots`. *Créer un constructeur suplémentaire. Penser à utiliser le mot clé* `super`. *Tester aussi les fonctions définies dans la classe* `Livre` *et redéfinir la fonction* `toString()`.

2 Exceptions et lecture/écriture dans un fichier

Lire http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-java/les-exceptions.

Exercice 5 *Créer une exception* `StockVideException`, *qui est envoyé lorsqu'on tente de* `vendre` *un livre pour lequel* `nbEnStock` *est négatif ou nul.*

Pour pouvoir traiter les entrées/sorties (console ou fichiers) il faut ajouter au tout début de votre fichier source : `import java.io.*;`. Il y a plusieurs façons d'ouvrir un fichier. Nous allons utiliser le mode caractère. La méthode suivante (à intégrer dans une classe) lit et affiche toutes les lignes du fichier `MonFichier.txt` :

```
public static void main(String[] args) throws IOException {
  File fichier = new File("monFichier.txt");
  Scanner sc = new Scanner (fichier);
  while (sc.hasNextLine())
  {
    String line = sc.nextLine();
    System.out.println (line);
  }
  sc.close();
}
```

Le code suivant écrit dans le fichier `output.txt` deux lignes de texte.

```
File fichierSortie = new File ("output.txt");
FileWriter fWriter = new FileWriter (fichierSortie);
PrintWriter pWriter = new PrintWriter (fWriter);
pWriter.println ("cette ligne est écrite dans le fichier");
pWriter.close();
```

Remarquez l'utilisation de l'instruction `throws IOException`. Il s'agit de la gestion des exceptions – des cas d'anomalies qui peuvent éventuellement se produire. Par exemple, le fichier auquel vous essayez d'accéder peut ne pas exister.
La suite de Conway est définie comme suit. Le premier terme est 1. Chaque terme de la suite se construit en *annonçant* le terme précédent :
$c_0 = 1$
$c_1 = 11$ (dans la suite précédente c_0 il y a *un 1*)
$c_2 = 21$ (dans la suite précédente c_1 il y a *deux 1*)
$c_3 = 1211$ (dans la suite précédente c_2 il y a *un deux* et *un 1*)

$c_4 = 111221$ (dans la suite précédente c_2 il y a *un 1*, *un 2* et *deux 1*)
...

Exercice 6 *Programmez une fonction qui lit dans un fichier* `conwayEntree` *un entier n et écrit dans un fichier* `conwaySortie` *les n premiers termes de la suite de Conway (un terme par ligne).*
Si le fichier d'entrée n'existe pas, imprimer le message `Fichier d'entrée inexistant`*. Même chose pour le fichier de sortie.*

3 Collections et Généricité

L'objectif de cet exercice est de comprendre la fonctionalité de genericité en Java qui permet à une classe de prendre en entrée un type. C'est ce qu'on utilise lorsqu'on crée ce qu'on appelle des `Collections`, par exemple des listes chaînées.

Exercice 7 *Créer une classe* `GrandNombre` *qui permet de stocker des nombres de taille arbitrairement grande. On utilisera une liste chaînée de chiffres décimaux. Implémenter les fonctions suivantes*

1. *(Constructeur)* `GrandNombre(int n)` *initialise la valeur du grand nombre à l'entier n.*
2. `addInt(int n)` *et* `multInt(int n)` *qui additionne et multiplie le grand nombre par un entier n et une fonction* `toString()` *qui renvoie une chaîne de caractères représentant le grand nombre.*
3. `setFactoriel(int n)` *qui stocke n! dans le grand nombre.*

Nous allons maintenant créer notre propre classe générique.

Exercice 8 *Créer une classe* `class ArbreBinaire<T>` *qui contient un arbre binaire dont les noeuds sont des objet de type* `T`*. Implémenter les méthodes* `int hauteur` *qui calcule la hauteur de l'arbre et* `parcoursLargeur`*,* `parcoursPrefixe` *qui affichent les éléments de l'arbre lors d'un parcours en largeur ou un parcours en profondeur dans l'ordre préfixe.*
Créer un arbre binaire contenant des entiers, et un arbre binaire contenant des livres.

Exercice 9 *On souhaite implémenter l'algorithme d'encodage de Huffman qui sert à la compression d'un texte donné. On suppose qu'on utilise l'alphabet latin sans accents (26 lettres donc).*

Algorithme 1: Algorithme Huffman

Input *: Un texte dans un fichier*
Output *: Un codage binaire de chaque lettre*
1 *Calculer le nombre d'apparitions de chaque lettre dans le texte;*
2 *Pour chaque caractère créer un arbre binaire avec un seul nœud étiqueté par la fréquence d'apparition de la lettre correspondante;*
3 *Itérativement, fusionner deux arbres dont la fréquence est minimale en rajoutant le même père aux deux racines. L'étiquette du père est la somme des étiquettes des fils;*
4 *L'algorithme s'arrête lorsqu'il ne reste qu'un seul arbre.*

Implémenter l'algorithme d'encodage de Huffman et afficher l'encodage du texte en entrée. L'encodage d'une lettre correspond au chemin dans l'arbre. Vous pourrez utiliser une `PriorityQueue` *de noeuds. Tester sur des exemples.*