Programmation Orientée Objet

Examen

3 heures - documents écrits autorisés
samedi 13 septembre 2003

Exercice 1 : Musée

Un musée regroupe et présente des *objets exposables*. Chaque objet appartient à une galerie (caractérisé par un numéro entier) et un descriptif (un texte sous forme d'une chaîne de caractères). Certains objets sont *artistiques*. Ils sont alors caractérisés par leur *art* et leur *auteur* (supposé unique et défini par une chaîne de caractère). On trouve également des objets définis comme *historiques*, une *période historique* leur est attribuée. Enfin on trouve des objets *fragiles* et des objets *rares* qui ont alors une *valeur estimée* (entière en euros).
Certains objets peuvent présenter plusieurs de ces caractéristiques : artistiques et rares comme des *tableaux de la renaissance* ; artistiques et historiques comme des *Vénus du paléolithique* (statuettes sculptées de la préhistoire) ; historiques, rares et fragiles comme des *rouleaux de parchemins étrusques* ; etc.
Le musée veut pouvoir disposer de la liste des objets de chaque type (liste des artistiques, liste des objets rares, liste des objets historiques, etc.). Il faut donc pouvoir distinguer chacune de ces catégories.
L'art d'une œuvre artistique est du type énuméré *Art* regroupant les valeurs : `peinture`, `sculpture`, `tapisserie` et `gravure`.
Une période historique a également sa valeur définie dans un type énuméré appelé *Periode*.

Q 1 . Représentez dans un diagramme de classes UML, les interfaces et classes, avec leurs attributs et méthodes pour modéliser les éléments décrits ci-dessus. Tous les termes en italiques dans le paragraphe précédent doivent apparaître que ce soit sous forme d'interface ou de classe ou d'attribut ou de méthode. Vous pouvez éventuellement en ajouter d'autres qui vous paraissent évidents bien que non mentionnés.
Il n'est pas besoin de faire apparaître une classe `Musee` dans le diagramme.
Vous ferez apparaître dans votre diagramme les liens d'impémentation et de dépendances entre les types.

Q 2 . Donnez le code java correspondant à `Art`.

Q 3 . On s'intéresse à une ébauche de la classe `Musee`. Donnez (uniquement) la signature des méthodes permettant d'ajouter :

1. un objet artistique à a liste des artistiques,
2. un objet rare à la liste des objets rares.

Exercice 2 : 421

Le jeu du 421 est un jeu de dés, qui se joue à au moins deux joueurs, 3 dés et 21 jetons. Le principe est de réaliser la meilleure combinaison possible avec les trois dés. En fonction de la combinaison gagnante, on attribuera des jetons au perdant. Selon les régions, les joueurs peuvent faire un ou plusieurs lancers à chaque tour.
Pour cet exercice, nous considérerons qu'il n'y a que deux joueurs, et que les joueurs n'ont droit qu'à un seul lancer par tour.
Le jeu du 421 se joue avec des jetons. Au cours de la première partie du jeu, nommée *la charge*, les joueurs vont se répartir les 21 jetons (initialement au milieu de la table). Le but sera d'en recevoir le moins possible. La phase de charge se termine quand il n'y a plus de jetons sur la table.
Dans la seconde partie du jeu, appelée la *décharge*, les joueurs tenteront de donner tous leurs jetons à leur adversaire. Un joueur est déclaré vainqueur s'il a réussi à ne prendre aucun jeton au cours de la charge ou dès qu'il n'en a plus au cours de la décharge. À chaque tour, c'est la meilleure combinaison qui détermine le nombre de jetons que prend le perdant (sur la table dans la phase de charge et à son adversaire dans la décharge).
Voici l'ordre des combinaisons et le nombre de jetons correspondant à chacune :

L'ordre des combinaisons est :

1. 421 (prononcer quatre vingt et un)
2. 3 as
3. 2 as - six
4. 3 six
5. 2 as - cinq
6. 3 cinq
7. 2 as - quatre
8. 3 quatre
9. 2 as - trois
10. 3 trois
11. 2 as - deux
12. 3 deux
13. six - cinq - quatre
14. cinq - quatre - trois
15. quatre - trois - deux
16. trois - deux - As

Les autres combinaisons sont, par ordre lexicographique décroissant, de 665 à 221.

Combinaison gagnante	nombre de jetons
421	10
3 as	7
2 as-six ou 3 six	6
2 as-cinq ou 3 cinq	5
...	...
une suite	2 jetons
autre	1 jeton

Implémentation en java

Q 1 . Définir une classe `De` dont les instances représentent des dés à jouer. Vous pourrez utiliser la classe `java.util.Random` pour simuler le lancer du dé. Cette classe permet de générer des valeurs de type primitif, de façon pseudo-aléatoire. Voici un exemple d'utilisation de cette classe pour générer des entiers :

```
java.util.Random hasard = new java.util.Random();
int i = hasard.nextInt(10) ; // renvoie un entier entre 0 et 9 ;
```

Q 2 . Nous allons maintenant écrire (partiellement) la classe `Lancer` dont les instances représentent un lancer de trois dés.

- Définir les attributs et constructeur(s) de cette classe.
- A chaque lancer, on veut connaître le dé le plus élevé, le dé intermédiaire, le dé le plus faible (ou du moins les valeurs de leurs faces).
- Ecrire une méthode `lancer` qui simule le lancer des 3 dés.
- Ecrire des méthodes à valeur booléennes permettant de tester si le lancer forme un 421, un brelan, une suite, possède deux-as (mais pas trois). ces méthodes seront utiles pour comparer deux lancers. Est-il nécessaire de les définir `public` ?

Q 3 . Pour le jeu, on a besoin de comparer deux lancers. La relation d'ordre sur les lancers est étroitement liée à la relation d'ordre sur les valeurs des dés. Modifier les classes `De` et classe `Lancer` pour qu'elles implémentent l'interface `Comparable` (voir Annexe).

Vous coderez la méthode `compareTo` de la classe `Lancer` de telle manière qu'elle renvoie le nombre de jetons que reçoit le lancer perdant : positif si le lancer `this` est gagnant, `0` si nul et négatif si `this` est perdant.

La **clareté** de la réponse à cette méthode sera prise en compte dans la correctiojn.

Q 4 . Définissez maintenant les classes `Joueur` et `Partie` permettant de simuler une partie de 421 à deux joueurs.

1 Annexe

description de l'interface `java.lang.Comparable`

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's natural ordering, and the class's compareTo method is referred to as its natural comparison method. Lists (and arrays) of objects that implement this interface can be sorted automatically by `Collections.sort` (and `Arrays.sort`). Objects that implement this interface can be used as keys in a sorted map or elements in a sorted set, without the need to specify a comparator.
The natural ordering for a class C is said to be consistent with equals if and only if `(e1.compareTo((Object)e2) == 0)` has the same boolean value as `e1.equals((Object)e2)` for every `e1` and `e2` of class `C`. Note that null is not an instance of any class, and `e.compareTo(null)` should throw a `NullPointerException` even though `e.equals(null)` returns false.
It is strongly recommended (though not required) that natural orderings be consistent with equals. This is so because sorted sets (and sorted maps) without explicit comparators behave "strangely" when they are used with elements (or keys) whose natural ordering is inconsistent with equals. In particular, such a sorted set (or sorted map) violates the general contract for set (or map), which is defined in terms of the equals method.
For example, if one adds two keys a and b such that `(!a.equals((Object)b) && a.compareTo((Object)b) == 0)` to a sorted set that does not use an explicit comparator, the second add operation returns false (and the size of the sorted set does not increase) because `a` and `b` are equivalent from the sorted set's perspective.
Virtually all Java core classes that implement comparable have natural orderings that are consistent with equals. One exception is `java.math.BigDecimal`, whose natural ordering equates `BigDecimal` objects with equal values and different precisions (such as `4.0` and `4.00`).
For the mathematically inclined, the relation that defines the natural ordering on a given class `C` is:
`{(x, y) such that x.compareTo((Object)y) <= 0}`.
The quotient for this total order is: `{(x, y) such that x.compareTo((Object)y) == 0}`.
It follows immediately from the contract for `compareTo` that the quotient is an equivalence relation on `C`, and that the natural ordering is a total order on `C`. When we say that a class's natural ordering is consistent with equals, we mean that the quotient for the natural ordering is the equivalence relation defined by the class's `equals(Object)` method: `{(x, y) such that x.equals((Object)y)}`.

méthode `compareTo`

L'interface `Comparable` contient une unique méthode : `public int compareTo(Object o)` telle que

- `o` est l'objet à comparer
- l'entier retourné est négatif (resp nul, positif) si l'objet courant est plus petit (resp égal, plus grand) que l'objet `o` passé en paramètre.
- Cette méthode peut retourner `ClassCastException` si `o` n'est pas d'un type qui permet de le comparer à l'objet courant.

Voici la documentation (javadoc) de cette méthode :

> Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
>
> In the foregoing description, the notation sgn(expression) designates the mathematical signum function, which is defined to return one of -1, 0, or 1 according to whether the value of expression is negative, zero or positive. The implementor must ensure
>
> `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`.
>
> (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)
>
> The implementor must also ensure that the relation is transitive:
>
> `(x.compareTo(y)>0 && y.compareTo(z)>0) implies x.compareTo(z)>0.`
>
> Finally, the implementer must ensure that
>
> `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all z.
>
> It is strongly recommended, but not strictly required that
>
> `(x.compareTo(y)==0) == (x.equals(y)).`
>
> Generally speaking, any class that implements the Comparable interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."