TD Exercices sur l'héritage

Exercice 1 : Gestion d'heures complémentaires

Chaque enseignant de l'université effectue un certain nombre d'heures d'enseignement dans une année. Suivant le statut de l'enseignant, un certain nombre de ces heures peut-être considéré comme *complémentaire*. Les heures complémentaires sont payées séparément à l'enseignant. Les volumes horaires sont exprimés en heures entières et le prix d'une heure complémentaire est de 35 Euros.

Le nom et le nombre d'heures total d'un enseignant sont fixés à sa création, puis seul le nom peut être librement consulté (méthode `nom()`).

D'autre part on veut pouvoir librement consulter un enseignant sur son volume d'heures complémentaires (méthode `hc()`) et sur la rétribution correspondante (méthode `retribution()`).

Il y a deux types d'enseignants :

les intervenants extérieurs : toutes les heures effectuées sont complémentaires,

les enseignants de la fac : seules les heures assurées au delà d'une charge statutaire de 192h sont complémentaires.

Q 1 . Modéliser les enseignants : quelles sont les classes ? où sont implémentées les méthodes ? lesquelles sont nouvelles, redéfinies?
Q 2 . Ecrire les classes Java.
Q 3 . Comment modifier le modèle pour y introduire les étudiants de troisième cycle qui assurent des enseignements : ce sont des enseignants extérieurs mais qui ne peuvent faire plus de 96 heures.
Q 4 . Les étudiants puisqu'ils n'ont pas d'employeur, voient leur rétribution diminuée de 18. Prendre en compte cette information pose-t-il un problème ?

Exercice 2 : Transport de marchandises

On souhaite modéliser en `java` le calcul de coûts de transport de marchandises. Les marchandises transportées seront des instances de la classe `Marchandise` dont le source java est donné figure 1.

```
public class Marchandise {
  private int poids ;
  private int volume ;
  public Marchandise (int poids, int volume) {
    this.poids = poids ;
    this.volume = volume ;
  }
  public int poids () {// retourne le poids en kg
    return poids ;
  }
  public int volume () {// retourne le volume en dm3
    return volume ;
  }
}
```

Figure 1: Le source java de la classe `Marchandise`

Les marchandises sont transportées sous la forme de cargaisons. Les seules fonctionnalités publiques des cargaisons sont :

ajouter qui permet d'ajouter une marchandise dans cette cargaison si cela est encore possible.

cout qui retourne, sous la forme d'un nombre entier d'euros, le coût total du transport de cette cargaison.

Une cargaison est par ailleurs également caractérisée par la distance sur laquelle elle est transportée. Ce renseignement est communiqué à la construction de la cargaison sous la forme d'un nombre entier de kilomètres. On précise qu'une cargaison ne peut réunir qu'un nombre limité de marchandises qui dépend d'un encombrement total de ces marchandises à ne pas dépasser. Cet encombrement est soit le poids total, soit le volume total des marchandises, selon le type de transport utilisé. Ce dernier influe aussi sur le calcul du coût de transport de la cargaison qui, de la même façon, dépent de l'encombrement des marchandises de la cargaison. On distingue donc

plusieurs types de cargaisons selon le moyen de transport utilisé. On peut cependant trouver une certain nombre de caractéristiques communes à toutes les cargaisons que vous devrez identifier. Les différents types de cargaison et leurs caractéristiques sont donnés par le tableau suivant :

type	encombrement	coût	limite
`Fluviale`	`poids`	`distance` $\times$ `encombrement`	`encombrement` ≤ 300000
`Routiere`	`poids`	$4 \times$ `distance` $\times$ `encombrement`	`encombrement` ≤ 38000
`Aerienne`	`volume`	$10 \times$ `distance` $\times$ `encombrement`	`encombrement` ≤ 80000
`AerienneUrgente`	`volume`	$2 \times$ le coût d'une cargaison `Aerienne`	`encombrement` ≤ 80000

Q 1 . Dessinez le graphe d'héritage concernant les différentes classes de cargaisons et écrivez ces classes en `java`.