Exercice d'application : gestion de l'heure

Enoncé
On souhaite disposer d'une classe permettant de gérer une heure exprimant une durée. Cette durée peut être exprimée

1. soit avec un nombre décimal (positif ou nul). Exemple : 4,51.
2. soit avec 3 entiers représentant respectivement les heures, les minutes et les secondes.
   Exemple : 4 heures 30 minutes 35 secondes.
   L'heure est ≥ 0, les minutes et les secondes ont des valeurs dans l'intervalle $[0, 59]$.
   Cette représentation est appelée `sexagésimale` (c-à-d fondée sur une numérotation de base soixante).

Des conversions entre les 2 modes de représentation doivent pouvoir être effectuées.
Cette classe sera nommée `SexDec`.
Sa structure interne est inconnue des utilisateurs c-à-d des autres classes du programme. A condition de ne pas changer le profil des méthodes, la structure interne peut être modifiée sans répercussion sur les autres classes du programme.

Dans la programmation une repésentation interne devra être choisie (l'une de ces 2 formes sera mémorisée) sans pour autant contraindre les utilisateurs qui manipuleront les heures sous les 2 formes.
La conversion entre les 2 représentations se fait comme suit :

1. Pour passer d'une valeur sexagésimale à une valeur décimale :
   `dec = heures + minutes/60. + secondes/3600.`
   Noter l'écriture `60.` et non `60` pour effectuer une division réelle et non entière.
2. Pour passer d'une valeur décimale à une valeur sexagésimale appliquer les calculs suivants :
   - `heures`
     Le nombre d'heures n'est rien d'autre que la partie entière de la durée décimale.
     Utiliser un `cast` sur un `int`.
     Exemple : `dec = 4.51` $\rightarrow$ `heures = 4`.
   - `minutes`
     En soustrayant le nombre d'heures de la durée décimale, on obtient un résidu d'au plus une heure qu'on convertit en minutes en le multipliant par 60.
     Garder la partie entière en faisant un nouveau `cast`.
     Exemple (suite) : `minDec = (dec-heures)*60 = (4.51-4)*60 = 30.6`
     à convertir en entier pour obtenir les minutes : `minutes = 30`.
   - `secondes`
     La valeur des secondes s'obtient de même.
     Soustraire des minutes décimales les minutes entières pour obtenir un résidu horaire correspondant à au plus une minute. A multiplier par 60 pour avoir la valeur finale. Garder la partie entière.
     Exemple (fin) : `secDec = (minDec-minutes)*60 = (30.6-30)*60 = 36.0`
     à convertir en entier pour avoir les secondes : `secondes = 36`.

Travail à effectuer

1. Ecrire la classe `SexDec` qui comporte un champ nommé `dec` de type `double` représentant la valeur décimale et la classe `TSexDec` qui comporte la méthode `main`. Cette première programmation doit être en violation avec le principe d'encapsulation. Réduire votre code au minimum.
   - Justifier pourquoi vous ne respectez pas le principe d'encapsulation.
   - Dire quelles sont les modifications à apporter pour encapsuler vos données.
2. Modifier la classe `SexDec` qui comporte à présent
   - Un champ (privé) nommé `dec` de type `double` représentant la valeur décimale.
   - Un constructeur recevant un argument de type `double` représentant une durée en heure décimale : `public SexDec(double heureDec)`.
   - Un accesseur `public double getDec()` fournissant l'heure décimale.
3. Expliquer pourquoi ce code est une illustration minimale du principe d'encapsulation.
   Dire comment le modifier.

4. Compléter la classe `SexDec` pour y ajouter
   - Un constructeur recevant trois arguments de type `int` représentant une valeur sexagésimale qu'on supposera normalisée (secondes et minutes entre 0 et 59, aucune limitation ne portera sur les heures).
     Son profil sera donc : `public SexDec(int heure, int mn, int sec)`.
   - Les 3 méthodes `public int getH()`, `public int getM()` et `public int getS()` fournissant respectivement les heures entières, les minutes et les secondes de cet objet.
5. Tester cette version avec un programme test de la forme :

```
import java.text.DecimalFormat;
public class TSexDec
{
   public static void main (String args[])
   {
      DecimalFormat df = new DecimalFormat("0.00");
      SexDec h1 = new SexDec(4.51) ;
      System.out.println("h1 - decimal = " + df.format(h1.getDec())
         + "  Sexa = " + h1.getH() + ":" + h1.getM() + ":" + h1.getS()) ;
      SexDec h2 = new SexDec(2, 32, 15) ;
      System.out.println("h2 - decimal = " + df.format(h2.getDec())
         + "  Sexa = " + h2.getH() + ":" + h2.getM() + ":" + h2.getS());
   }
}
```

   qui lors de l'exécution affiche :

```
h1 - decimal = 4,51  Sexa = 4:30:35
h2 - decimal = 2,54  Sexa = 2:32:15
```

6. Pour une raison quelconque (question d'efficacité ou autre), il s'avère préférable d'utiliser la représentation sexagésimale. Avant d'entreprendre sa programmation, veuillez prendre le temps pour effectuer la réflexion et le travail préliminaire suivants :
   (a) Dire quelles sont les modifications à entreprendre
      - sur le nom de la classe
      - sur les attributs
      - sur les constructeurs et méthodes (profil, corps et nombre)
      - sur la classe `TSexDec` représentant toutes les classes clientes.

   (b) Un problème supplémentaire se présente si vous désirez garder la première version de `SexDec` pour vos archives personnelles. Pour qu'il n'y ait pas de confusion entre les 2 versions, vous devez rendre cette permière version inaccessible.
   Vous pouvez, par exemple
      - ranger la classe (code source et Java Byte Code) dans une autre catalogue,
      - mettre le code en commentaire.
7. Ecrire le code de cette classe avec la représentation sexagésimale.
8. Tester cette version avec le même programme test `TSexDec`.

Erreurs à ne pas commettre
L'exercice est mal compris et réalisé si vous vous trouvez dans l'un des cas suivants :

1. Vous avez dans la classe `SexDec` un mixte des attributs. Exemples d'erreur :
   - avoir simultanément les 4 attributs dans la classe
   - avoir une heure déciminale + minutes + secondes
2. Avoir dans la classe `SexDec` les constructeurs et méthodes uniquement liés à la structure interne. Exemple :
   - représentation décimale et constructeur `SexDec(double)` avec la seule méthode `getDec()`,
   - représentation sexagésimale avec le constructeur `SexDec(int,int,int)` et les 3 méthodes `getH()`, `getM()` et `getS()`
3. Renommer la classe entre les 2 versions. Exemple `SexDec` dans la première version et `SexDec1` dans la seconde ce qui nécessite de modifier `TSexDec`.