

6 avril 2011

Contrôle d'informatique no 3

Durée : 1 heure 45'

Nom :

Groupe :

Prénom :

No sujet	1	2	3	Bonus
Nombre points	41 points	40 points	39 points	10 points

Remarque générale : toutes les questions qui suivent se réfèrent au langage de programmation Java (à partir du JDK 5.0) et les réponses doivent être rédigées à l'encre et d'une manière propre sur ces feuilles agrafées.

Remarques initiales :

- Il est conseillé de lire les sujets jusqu'à la fin, avant de commencer la rédaction de la solution.
- Les trois sujets peuvent être résolus (éventuellement) séparément, mais après avoir lu et compris l'ensemble des énoncés.
- En Java, le nombre π est disponible sous la forme du champ statique et final **PI** de la classe `java.lang.Math`.
- L'opérateur % est l'opérateur *modulo*.
- Le périmètre d'un cercle de rayon r vaut $2 \cdot \pi \cdot r$.
- Le périmètre d'un secteur circulaire de rayon r et d'angle au centre α (en degrés) vaut

$$2 \cdot r + \frac{\alpha}{180} \cdot \pi \cdot r$$

- Une question "bonus" se trouve à la fin du contrôle.

On considère un projet Java qui contient 4 classes regroupées dans un package nommé *cms_ctr3*, à savoir :

- la classe abstraite *Ovale* dont l'en-tête et le corps sont indiqués ci-dessous ;
- la classe non abstraite *Cercle* qui dérive de la classe de base *Ovale* et que vous devez définir en fonction des consignes précisées au point 1 ;
- la classe non abstraite *Secteur* qui dérive de la classe de base *Cercle* et que vous devez définir en fonction des consignes précisées au point 2 ;
- la classe principale *CP_Ctr3Exo1* dont le code source est donné au point 3 et vous devrez indiquer quels seront les résultats affichés dans la fenêtre console suite à son exécution.

Voilà le code source de la classe abstraite *Ovale* :

```
package cms_ctr3;

public abstract class Ovalle
{
    public abstract double calculerPerimetre();
    public abstract int ordonner(Ovalle ov);
} //fin de la classe Ovalle
```

Vous ne devez rien modifier dans la classe *Ovale* (ni dans son en-tête ni dans son corps). Cette classe sera la classe mère de la classe *Cercle* définie au point 1 (et la classe "grand-mère" de la classe *Secteur* définie au point 2).

1. Le but de ce premier sujet est d'écrire le code source de la classe *Cercle* qui permet l'instanciation d'objets correspondant à des cercles précisés par un nom et un rayon (exprimé en mètres).

Plus précisément, la classe *Cercle* fait partie du package *cms_ctr3*, est publique et dérive directement de la classe *Ovale* (qui fait partie du même package). Dans le corps de la classe *Cercle*, on définit :

- un champ privé *nom* de type chaîne de caractères, avec la valeur initiale explicite *sansNom* (et le champ *nom* ne peut pas correspondre à une adresse non définie ou à une chaîne vide ou à une chaîne contenant au moins une espace) ;
- un champ privé *rayon* de type nombre réel, avec la valeur initiale explicite *1* (et le champ *rayon* doit être strictement positif) ;

- une méthode privée d'instance **verifier** qui a un argument de type chaîne de caractères et qui retourne une valeur de type logique, à savoir :
 - **faux** si l'argument de la méthode est une adresse non définie (c'est-à-dire si sa valeur est **null**) ou l'adresse d'une chaîne vide ou l'adresse d'une chaîne qui contient au moins un espace ;
 - **vrai** dans tous les autres cas ;
- une méthode publique « setter » appelée **setNom** qui permet la modification de la valeur du champ privé **nom** ; cette méthode doit respecter les consignes suivantes :
 - si la méthode **verifier** appelée avec le même argument que celui de la méthode **setNom** retourne la valeur logique **vrai**, on stocke dans le champ **nom** la valeur de l'argument de la méthode **setNom** ;
 - autrement, on affiche le message **Nom du cercle non valide !** dans la fenêtre console et on stocke dans le champ **nom** la chaîne de caractères **parDefaut** ;
- une méthode publique « getter » appelée **getNom** qui retourne (sans aucune vérification) la valeur du champ privé **nom** ;
- une méthode publique « setter » nommée **setRayon** qui permet la modification de la valeur du champ privé **rayon** ; cette méthode doit respecter les consignes suivantes :
 - si la valeur de son argument est strictement positive, on stocke dans le champ **rayon** la valeur de l'argument de la méthode ;
 - autrement, on affiche le message **Rayon non valide !** dans la fenêtre console et on stocke dans le champ **rayon** la valeur **10** ;
- une méthode publique « getter » nommée **getRayon** qui retourne (sans aucune vérification) la valeur du champ privé **rayon** ;
- un constructeur public sans argument qui affiche simplement le message **Cercle sans argument !** dans la fenêtre console ;
- un constructeur public avec deux arguments, le premier de type chaîne de caractère et le deuxième de type numérique réel, et qui doit respecter les consignes suivantes :
 - la valeur du premier argument « est stockée » dans le champ **nom** par un appel à la méthode « setter » adéquate ;
 - la valeur du deuxième argument « est stockée » dans le champ **rayon** par un appel à la méthode « setter » adéquate ;
 - le message **Cercle avec deux arguments !** est affiché dans la fenêtre console.

De plus, dans la classe dérivée **Cercle**, on **redéfinit** :

- la méthode héritée **calculerPerimetre** qui retourne la valeur du périmètre du cercle correspondant à l'objet appelant la méthode ;
- la méthode héritée **ordonner** qui, après avoir affiché le message « **On ordonne des ovales !** » dans la fenêtre console, retourne la valeur entière :
 - o **1** si le périmètre correspondant à l'objet appelant la méthode est strictement plus grand que le périmètre correspondant à l'objet argument de la méthode ;
 - o **0** si le périmètre correspondant à l'objet appelant la méthode est égal au périmètre correspondant à l'objet argument de la méthode ;
 - o **-1** si le périmètre correspondant à l'objet appelant la méthode est strictement plus petit que le périmètre correspondant à l'objet argument de la méthode ;
- la méthode **toString** (héritée de la classe ascendante **java.lang.Object**) qui est précisée plus loin et qui n'a pas besoin d'être écrite par vous.

On présente ci-dessous le squelette de la classe **Cercle** et il faut compléter ce canevas en fonction des indications données en commentaire.

```
//déclaration de package
.....

.....

//en-tête de la classe Cercle
.....

.....

{
    //déclarations et initialisations des champs
    .....

    .....

    //définition de la méthode verifier
    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....
```

```
//définition de la méthode setNom
```

```
//définition de la méthode getNom
```

```
//définition de la méthode setRayon
```

//définition de la méthode **getRayon**

//définition du **constructeur sans argument**

//définition du **constructeur avec deux arguments**

//redéfinition de la méthode **calculerPerimetre**

//redéfinition de la méthode **ordonner**

```

.....
.....
.....
.....
.....
.....
.....
.....
.....
//redéfinition de la méthode toString
//rien à ajouter ci-dessous
@Override
public String toString()
{
    return "Cercle " + nom + " de rayon " + rayon + ".";
}
}
} //fin de la classe Cercle

```

2. Le but de ce deuxième sujet est d'écrire le code source de la classe **Secteur** qui permet l'instanciation d'objets correspondant à des secteurs circulaires précisés par un nom, un rayon (exprimé en mètres) et un angle au centre (exprimé en degrés).

La classe **Secteur** fait partie du package **cms_ctr3**, est publique et dérive (directement) de la classe de base **Cercle** (qui fait partie du même package). Dans le corps de la classe **Secteur**, on définit :

- un champ (propre) privé **angle** de type nombre réel avec la valeur initiale explicite de 30 (et le champ **angle** doit être compris dans l'intervalle]0; 360]) ;
- une méthode publique « setter » nommée **setAngle** qui permet la modification de la valeur du champ privé **angle** ; cette méthode doit respecter les consignes suivantes :
 - o si la valeur de son argument est strictement positive et plus petite ou égale à 360, on stocke dans le champ **angle** la valeur de l'argument de la méthode ;
 - o autrement, on affiche le message **Angle non valide** dans la fenêtre console et on stocke dans le champ **angle** la valeur **45** ;
- une méthode publique « getter » nommée **getAngle** qui retourne (sans aucune vérification) la valeur du champ privé **angle** ;
- un constructeur public sans argument qui affiche simplement le message **Secteur sans argument !** dans la fenêtre console ;

- un constructeur public avec un argument de type numérique réel qui "stocke" la valeur de son argument dans le champ propre *angle* par un appel à la méthode « setter » adéquate et affiche le message *Secteur avec un argument !* dans la fenêtre console ;
- un constructeur public avec trois arguments, le premier argument de type chaîne de caractères et les deux autres arguments de types numériques réels, et qui doit respecter les consignes suivantes :
 - les valeurs du premier et du deuxième argument « sont stockées » dans les champs hérités *nom* et, respectivement, *rayon* par un appel explicite au constructeur de la classe de base ;
 - la valeur du troisième argument « est stockée » dans le champ propre *angle* par un appel à la méthode « setter » adéquate ;
 - le message *Secteur avec trois arguments !* est affiché dans la fenêtre console.

De plus, dans la classe dérivée *Secteur*, on **redéfinit** :

- la méthode héritée *calculerPerimetre* qui doit respecter les consignes suivantes :
 - si la valeur du champ propre *angle* (de l'objet appelant la méthode) est **360**, la méthode retourne la valeur du périmètre d'un cercle obtenue par un appel à la méthode (d'origine) homonyme de la classe mère *Cercle* ;
 - autrement, la méthode retourne la valeur du périmètre du secteur circulaire correspondant à l'objet appelant la méthode ;
- la méthode héritée *ordonner* qui doit respecter les consignes suivantes :
 - si l'argument de la méthode est de type *Secteur* (ou compatible par polymorphisme), la méthode affiche le message *On ordonne des secteurs circulaires !* et retourne la valeur entière :
 - **101** si l'angle au centre du secteur circulaire correspondant à l'objet appelant la méthode est strictement plus grand que l'angle au centre du secteur circulaire correspondant à l'objet argument de la méthode ;
 - **100** si l'angle au centre du secteur circulaire correspondant à l'objet appelant la méthode est égal à l'angle au centre du secteur circulaire correspondant à l'objet argument de la méthode ;
 - **99** si l'angle au centre du secteur circulaire correspondant à l'objet appelant la méthode est strictement plus petit que l'angle au centre du secteur circulaire correspondant à l'objet argument de la méthode ;

- autrement, la méthode affiche le message ***On ordonne un secteur et un ovale !*** et retourne la valeur obtenue par un appel à la méthode (d'origine) homonyme de la classe mère ***Cercle*** ;
- la méthode héritée ***toString*** qui est précisée plus loin et qui n'a pas besoin d'être écrite par vous.

On présente ci-dessous le squelette de la classe ***Secteur*** et il faut compléter ce canevas en fonction des indications données en commentaire.

```
//déclaration de package
.....

.....

//en-tête de la classe Secteur
.....

.....

{
    //déclaration et initialisation du champ propre
    .....

    .....

    //définition de la méthode setAngle
    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    //définition de la méthode getAngle
    .....

    .....

    .....

    .....
```

```
//définition du constructeur sans argument
```

```
//définition du constructeur avec un argument
```

```
//définition du constructeur avec trois arguments
```

```
//redéfinition de la méthode calculerPerimetre
```


[illegible]

[illegible]

Bonus

Soit le programme suivant :

```
package cms_ctr3;

interface IFaisable
{
    void faire();
} //fin de l'interface IFaisable

interface IRefaisable extends IFaisable {
    void refaire();
} //fin de l'interface IRefaisable

class Travail implements IRefaisable
{
    public void travailler(){ }
    public void faire(){ }
    public void refaire() { }
} //fin de la classe Travail

public class CP_Ctr3Bonus
{
    public static void main(String[] args)
    {
        IFaisable ref = new Travail();

        //ici on introduit une des instructions proposées
        //ci-dessous

    } //fin de la méthode main
} //fin de la classe principale
```

Quelles instructions introduites **séparément** à la place du commentaire ci-dessus, produisent une erreur (à la compilation) ? Choisir et encercler les lettres correspondant aux réponses qui conviennent.

- a. ref.travailler();
- b. ref.faire();
- c. ref.refaire();
- d. (Travail)ref.travailler();
- e. ((Travail)ref).travailler();
- f. ((Travail)ref).faire();
- g. ((Travail)ref).refaire();
- h. (IRefaisable)ref.travailler();
- i. ((IRefaisable)ref).travailler();
- j. ((IRefaisable)ref).refaire();