Héritage, sous-typage, redéfinition

Exercice 1 - One Ring for ...

Le but de cet exercice est de construire un anneau comme étant un cercle dont on a évidé une zone circulaire définie par son rayon interne.
Préliminaire : avant d'écrire la classe `Ring`, Ajouter à la classe `Circle` du TD précédent la méthode `surface()` qui renvoie la surface d'un cercle (en fait du disque).
Modifier la méthode `toString()` pour qu'elle affiche la surface.
Rappeler dans un premier temps, dans quel cas il est judicieux de faire de l'héritage.
Écrire la classe `Ring` qui hérite de la classe `Circle`.

1 Écrire un constructeur de la classe `Ring` prenant en paramètre, un centre, un rayon et un rayon interne. Faîte attention à ce que le rayon interne soit inférieur au rayon de l'anneau.
2 Écrire la méthode `equals()` qui test l'égalité de deux anneaux.
3 Écrire la méthode `toString()` qui affiche le centre, le rayon et le rayon interne.

Quelle est le problème avec la méthode `surface` ?

```
Ring ring=new Ring(new Point(1,2),2,1);
System.out.println(ring.surface());
```

Résoudre le problème posé par la méthode ?
Ensuite,

1 Implanter une méthode `contains(Point)`. Quels problèmes cela pose ?
PS: il existe deux solutions dont une plus élégante que l'autre.
2 Ecrire la méthode `contains(Point p,Ring... rings)` qui renvoie vrai si un point est contenu dans un des anneaux.

Exercice 2 - Évaluateur d'expression

Le but de cet exercice est de construire un évaluateur d'expressions arithmétiques simples. Ces expressions sont représentées sous forme d'arbre.
On veut un type commun `Expr` représentant des expressions arithmétiques qui peuvent être soit une valeur réelle (de type `Value`) soit une opération d'addition (de type `Add`) qui permet d'effectuee l'addition de deux expression.
On veut de plus être capable d'évaluer de ces expressions en utilisant la méthode `eval()`.

```
Expr value=new Value(7.0);
System.out.println(value.eval()); // affiche 7.0
Expr add=new Add(new Value(3.0),value);
System.out.println(add.eval());   // affiche 10.0
Expr expr=new Add(new Value(2.0),add);
System.out.println(expr.eval());  // affiche 12.0
```

1 Écrire les trois classes `Expr`, `Value` et `Add` et leurs méthodes `double eval()`.
2 Implanter la méthode `toString()` sur les différentes expressions.
3 Discuter sur le fait de transformer la classe `Expr` en classe abstraite ou en interface. Faîtes les changements qui s'imposent.
4 Ajouter les classes `Mult` et `Divide` permettant d'effectuer respectivement la

multiplication et la division. Refactoriser le code pour mettre à un seul endroit les codes communs.

5 On souhaite modifier le programme pour qu'il évalue une expression pris sur la ligne de commande :

```
Java Evaluator "+" "7" "*" "3" "2"
=13
```

Chaque chaîne de caractère est analysée pour savoir s'il s'agit d'un opérateur où d'une valeur. L'expression est construite par un parcours récursif, puis évaluée.

6 Modifier le code pour que si la ligne de commande n'ait pas d'argument, les expressions soit prisent sur l'entrée sandard.
PS : essayer de partager le code au maximum avec les fonctions déjà existantes.