POO – TD/TP 3

Exercice 1 – Interfaces

1. Écrivez l'interface `StringFilter` contenant l'unique méthode `String filter(String s)`. Les implémentations de cette méthode doivent transformer la chaîne `s` puis de retourner le résultat de cette transformation.
2. Écrivez les classes suivantes implémentant l'interface `StringFilter` :
   - `UpperCaseStringFilter` convertit les caractères de s en majuscules.
   - `LowerCaseStringFilter` convertit les caractères de s en minuscules.
   - `PrefixStringFilter` conserve les `n` premiers caractères de `s`. La valeur de `n` est fournie lors de la construction d'une instance de la classe.
   - `PostfixStringFilter` conserve les `n` derniers caractères de `s`.
   - `AsciiStringFilter` conserve les caractères dont le code est inférieur à 128.
   - `UnaccentStringFilter` supprime les accents.
3. Écrivez la méthode statique `String[] filter(String[] strings, StringFilter filter)` qui applique le filtre aux chaînes du tableau `strings` et qui retourne un tableau contenant les chaînes transformées.
4. Écrivez la classe `CompositeStringFilter` qui implémente l'interface `StringFilter` et qui applique successivement sur la chaîne `s` les filtres du tableau `StringFilter[] filters` passé au constructeur.

Exercice 2 – Formules

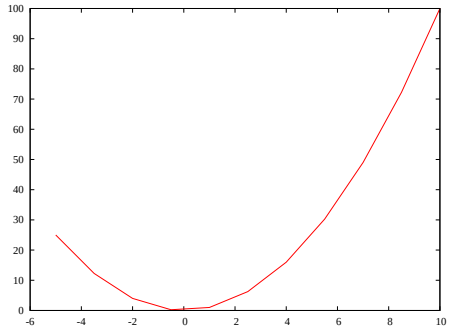
1. Décrivez l'interface `Formula` et écrivez les classes `Variable`, `Sum` et `Product` de façon à obtenir le comportement suivant :

```
Variable x = new Variable("x", 2.5);
Variable y = new Variable("y", 4);
Formula formula =
          new Sum(x, new Product(y, new Sum(x, y)));
System.out.println(formula.asString()); // "(x+(y*(x+y))"
System.out.println(formula.asValue()); // "28.5"
x.set(5);
System.out.println(formula.asValue()); // "41.0"
```

2. Modifiez les classes `Sum` et `Product` de façon à réaliser les opérations sur un ensemble de formules. Ces formules sont passées au constructeur sous la forme d'un tableau. Faites en sorte que le programme proposé à la question 1 fonctionne après la modification.
3. Ajoutez les classes suivantes :
   - `AbsoluteValue` : valeur absolue "$|f|$".
   - `Square` : carré "f^2".
   - `SquareRoot` : racine carrée "$\sqrt{f}$".
   - `Power` : puissance "f^k".
   - `Minimum` : minimum "$\min(f_1, f_2, ..., f_k)$".
   - `Maximum` : maximum "$\max(f_1, f_2, ..., f_k)$".
4. En TP, écrivez la méthode statique `void generatePoints(Formule formula, Variable variable, double startValue, double endValue, double step)` qui affiche les valeurs de `formula` en faisant varier `variable` de `startValue` à `endValue` en ajoutant la valeur de `step` à chaque itération.
5. Obtenez la courbe suivante à l'aide des informations ci-dessous :

```
Variable variable = new Variable("variable", 0);
Formule formula = new Square(variable);
generatePoints(formula, variable, -5, 10, 1.5);
```

```
-5.0 25.0
-3.5 12.25
-2.0 4.0
-0.5 0.25
1.0 1.0
2.5 6.25
4.0 16.0
5.5 30.25
7.0 49.0
8.5 72.25
10.0 100.0
```




```
$ java FormulaMain > data.txt
$ gnuplot
gnuplot> plot "data.txt" with lines
```

Exercice 3 – Formes

1. Écrivez une classe `Point` qui contient deux entiers `x` et `y` représentant les coordonnées du point. La classe fournit un unique constructeur prenant en paramètre les coordonnées du point. La classe propose également la méthode `double distance(Point p)` qui calcule la distance euclidienne entre le point `p` et le point sur lequel est appelée la méthode (`this`), c'est-à-dire, $\sqrt{(\text{p.x} - \text{this.x})^2 + (\text{p.y} - \text{this.y})^2}$.

2. Écrivez une classe `Rectangle` contenant deux points `p1` et `p2` (ces deux points sont donnés au constructeur). Les points `p1` et `p2` représentent deux points opposés du rectangle. La classe propose les méthodes :
   — `boolean contains(Point point)` qui retourne `true` si `point` est dans le rectangle.
   — `int getPerimeter()` qui retourne le périmètre du rectangle.

3. Écrivez une classe `Main` contenant la méthode `main` et une méthode statique `double getSumOfPerimeters( Rectangle[] rectangles, Point point)` retournant la somme des périmètres des rectangles de `rectangles` qui contiennent le point `point`.

4. Écrivez une classe `Circle` contenant deux points `center` et `point` (ces deux points sont donnés au constructeur). Le point **center** représente le centre du cercle et le point `point` un point sur le périmètre du cercle. La classe propose les méthodes :
   — `double radius()` qui retourne le rayon du cercle.
   — `boolean contains(Point point)` qui retourne `true` si `point` est dans le cercle.
   — `double getPerimeter()` qui retourne le périmètre du cercle.

5. Écrivez une interface `Shape` qui définit les deux méthodes `boolean contains(Point p)` et `double getPerimeter()`. Modifiez les classes `Rectangle` et `Circle` de sorte qu'elles implémentent cette interface.

6. Dans la classe `Main`, écrivez une méthode statique `double getSumOfPerimeters(Shape[] shapes, Point point)` retournant la somme des périmètres des formes de `shapes` qui contiennent le point `point`. Peut-elle (ou doit-elle) remplacer celle écrite à la question 3 ?