PC – TD/TP 1

Vous devez terminer l'implémentation des programmes en dehors des TP

Exercice 1 – Vecteur d'entiers (en TD et TP)

Écrivez la classe `Vector` qui permet de gérer un tableau dont la capacité augmente automatiquement si celui-ci est plein. Cette classe contient deux attributs : un tableau d'entiers `array` et un entier `size`. La longueur du tableau `array` peut être supérieure à `size`. Néanmoins, les entiers réellement présents dans le vecteur sont stockés dans les `size` premières cases du tableau `array`. Le constructeur prend en paramètre la longueur initiale du tableau `array`, c'est-à-dire la capacité initiale du vecteur. Si la longueur du tableau `array` ne permet plus de conserver tous les éléments du vecteur, elle est automatiquement augmentée à l'aide de la méthode `ensureCapacity`. La classe fournit les méthodes suivantes :

- `void ensureCapacity(int capacity)` fait en sorte que le tableau `array` puisse contenir `capacity` éléments. Si la capacité actuelle du vecteur est inférieure à `capacity`, la capacité est augmentée. La nouvelle capacité doit être égale à max(capacity, $2 \times$ capacité actuelle). Les nouvelles cases du tableau `array` sont initialisées à zéro. Le nombre d'éléments (c'est-à-dire la valeur de `size`) n'est pas modifié.
- `void resize(int size)` modifie la taille du vecteur. Si la capacité est inférieure à `size`, elle est augmentée. Dans tous les cas les nouvelles cases sont initialisées à zéro.
- `int size()` retourne la taille actuelle du vecteur.
- `boolean isEmpty()` retourne `true` si le vecteur est vide, `false` sinon.
- `void add(int value)` ajoute l'entier `value` à la fin du vecteur.
- `void set(int index, int value)` affecte l'élément `value` à la position `index` dans le tableau. Si le tableau contient moins de `index+1` éléments, la méthode ne fait rien.
- `int get(int index)` retourne l'élément à la position `index` dans le vecteur. Si le vecteur contient moins de `index+1` éléments, la méthode retourne `0`.

Exercice 2 – Pile d'entiers (uniquement en TP)

Écrivez la classe `Stack` qui gère une pile d'entiers. Elle contient un vecteur d'entiers (écrit à l'exercice précédent) et fournit les méthodes suivantes :

- `void push(int value)` empile l'entier `value`.
- `int peek()` retourne l'entier en haut de la pile (sans le dépiler).
- `int pop()` dépile l'entier en haut de la pile et le retourne.
- `int size()` retourne le nombre d'entiers dans la pile.
- `boolean isEmpty()` retourne `true` si la pile est vide, `false` sinon.

Exercice 3 – Interfaces (début en TD, finir en TP)

1. Écrivez l'interface `StringFilter` contenant l'unique méthode `String filter(String string)`. Les implémentations de la méthode `filter` doivent transformer la chaîne `string` puis de retourner le résultat de cette transformation.
2. Écrivez les classes suivantes implémentant l'interface `StringFilter` :
   - `UpperCaseStringFilter` convertit en majuscules les caractères.
   - `LowerCaseStringFilter` convertit en minuscules les caractères.
   - `PrefixStringFilter` conserve les `n` premiers caractères de `string`. La valeur de `n` est fournie lors de la construction d'une instance de la classe.
   - `PostfixStringFilter` conserve les `n` derniers caractères de `string`.
   - `AsciiStringFilter` conserve les caractères de code inférieur à 128.
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 `string` les filtres du tableau `StringFilter[] filters` passé au constructeur.

Exercice 4 – 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(Formula 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.

   Nous voulons que le code suivant...

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

...produise la sortie suivante...

```
-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
```

5. En TP, utilisez `gnuplot` pour visualiser les données obtenues à la question 4 en tapant les commandes suivantes dans une console :

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