POO – TD/TP 2

Exercice 1 – Points

1. Écrivez une classe `Point` qui représente un point sur le plan. Elle possède deux attributs flottants `x` et `y`.
2. Ajouter un constructeur afin de permettre l'initialisation de `x` et de `y`.
3. Ajouter une méthode `double distance()` qui retourne la distance du point à l'origine du repère.
4. Ajouter une méthode `double distance(Point point)` qui calcule la distance du point "`this`" au point `point`.
5. Ajouter une méthode `void translate(double dx, double dy)` qui translate le point de `dx` sur les abscisses et de `dy` sur les ordonnées.
6. Ajouter une méthode `Point isobarycentre(Point[] points)` qui retourne l'isobarycentre d'un ensemble de points. Est-ce que cette méthode peut être statique ?
7. Écrire une classe `Segment` qui possède deux attributs `point1` et `point2` de type `Point` représentant les deux extrémités du segment.
8. Ajouter une méthode `int length()` qui calcule et retourne la longueur du segment. Est-ce qu'une partie du calcul peut être déléguée à une méthode de la classe `Point` ?

Exercice 2 – Listes chaînées

Une liste chaînée est une suite de maillon. Chaque maillon est connecté au maillon qui le précède dans la liste (s'il existe) ainsi qu'au maillon qui le suit dans la liste (s'il existe). Ici, chaque maillon va également contenir un entier.

1. Écrivez une classe `Node` représentant un maillon d'une liste.
2. Ajouter un constructeur qui permet de connecter le maillon et d'initialiser la valeur qu'il contient.
3. Écrivez une classe `List` qui possède un attribut contenant la tête de la liste ou `null` si la liste ne possède pas de maillon.
4. Ajouter une méthode `void addFirst(int value)` qui ajoute un maillon contenant la valeur `value` en tête de liste.
5. Ajouter une méthode `void addLast(int value)` qui ajoute un maillon contenant la valeur `value` en queue de liste.
6. Ajouter une méthode `boolean contains(int value)` qui retourne `true` si la liste contient la valeur `value` et `false` dans le cas contraire.
7. Ajouter une méthode `void remove(int value)` qui permet de supprimer le premier maillon de la liste qui contient la valeur `value`.

Exercice 3 – Arbres binaires

Un arbre binaire est une structure de données. Chaque noeud de la structure, à l'exception de la racine de la structure possède un père. Par conséquent, chaque noeud possède une liste d'enfants potentiellement vide. Ici, chaque noeud va contenir une chaîne de caractères comme information supplémentaire.

1. Écrivez une classe `Node` représentant un noeud de l'arbre. Vous devez faire en sorte que chaque instance de cette classe contienne les références du père et des enfants du noeud.
2. Ajouter un constructeur avec un unique paramètre permettant d'initialiser la chaîne de caractères. Après sa construction, un noeud ne possède ni père ni enfant.
3. Ajouter une méthode privée `void setParent(Node node)` qui permet d'attribuer un père au noeud.
4. Ajouter une méthode `void addChild(Node child)` permettant d'ajouter un fils au noeud.
5. Écrire `void printDescendants()` et `void printAncestors()` qui affichent les descendants et ancêtres d'un noeud :

```
Node root = new Node("root");
Node a = Node("a"); Node b = Node("b");
Node c = Node("c"); Node d = Node("d");
root.addChild(a); root.addChild(b);
b.addChild(c); b.addChild(d);
root.printDescendants(); // -> a b c d
d.printAncestors(); // -> b root
```

Exercice 4 – Tableau dynamique

Écrivez la classe `Vector` qui gère un tableau dont la capacité varie dynamiquement. Elle contient deux champs : un tableau d'entiers `array` et un entier `size`. Les entiers présents dans le vecteur sont les `size` premiers éléments du tableau `array`. Le constructeur prend en paramètre la taille initiale du tableau `array` (c'est-à-dire la capacité initiale du vecteur). Si la taille du tableau `array` ne permet plus de conserver les `size` éléments du vecteur, elle est automatiquement augmentée à l'aide de la méthode `ensureCapacity`. La classe fournit les méthodes suivantes :

— `void ensureCapacity(int capacity)` assure une contenance de `capacity` éléments. Si la capacité actuelle du vecteur est inférieure à `capacity`, la capacité est augmentée. La nouvelle capacité doit être égale à $\max(capacity, 2 \times \text{capacité actuelle})$. 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 `s`, elle est augmentée et 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 element)` ajoute un élément à la fin du vecteur.

— `void set(int index, int element)` modifie l'élément à la position `i` dans le tableau. Si le tableau contient moins de `i+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`.

En TP, n'oubliez pas de tester les méthodes de votre classe.

Exercice 5 – Pile

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 element)` empile un nouvel entier.

— `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 empty()` retourne `true` si la pile est vide, `false` sinon.