

Application de la classe Point

Question 1 : D'après vous quel modificateur d'accès `public` ou `private` doit-on utiliser pour les attributs, méthodes et constructeurs de la classe `PolygonalChain` ?

- On applique le principe d'encapsulation et on préfère donc rendre privés les attributs d'une classe.
 - On utilisera donc `private` pour `Point[] vertices`
- Généralement, les constructeurs sont publics sauf raisons spécifiques. Ici, il n'y a pas de raisons de rendre privés les constructeurs.
 - On utilisera donc `public` pour `PolygonalChain(int size)`.
 - On utilisera donc `public` pour `PolygonalChain(Point [] vertices)`.
- Généralement, les méthodes sont publics sauf raisons spécifiques. Ici, il n'y a pas de raisons de rendre privés les méthodes.
 - On utilisera donc `public` pour `Point getVertex(int index)`.
 - On utilisera donc `public` pour `void setVertex(int index, Point p)`.
 - On utilisera donc `public` pour `String toString()`.
 - On utilisera donc `public` pour `double length()`.
 - On utilisera donc `public` pour `boolean isClosed()`.

Question 2 : Donnez le code de `PolygonalChain(int size)`.

```
public PolygonalChain(int size){  
    this.vertices = new Point[size];  
}
```

Question 3 : Donnez le code de `toString()`.

```
public String toString(){  
    String result = "";  
    result += "[";  
    for(int index=0; index<vertices.length; index++){  
        result += vertices[index];  
    }  
}
```

```

    if (index != vertices.length-1) {
        result += ", ";
    }
}
result += "]";
return result;
}

```

Question 4 : Quel est l’affichage produit par le code ci-dessus ?

Le code affiche :

`[(1,1), (3,3)]`

En effet, l’attribut `vertices` de `polygonalChain` et la variable `points` pointent toutes les deux vers le même tableau. En modifiant le tableau `points`, on modifie donc le tableau `vertices` de `polygonalChain`.

Question 5 : Est-ce que cela vous paraît être un comportement normal ? Comment peut-on le corriger ?

Non, ce n’est pas un comportement normal. Cela contrevient au principe d’encapsulation. Même en mettant l’attribut `vertices` en privé, l’utilisateur de la classe `PolygonalChain` peut toujours le modifier sans passer par une méthode (et donc sans contrôle). Une manière de remédier à cela est de faire une copie du tableau donné en argument du constructeur.

```

public PolygonalChain(Point[] vertices) {
    this.vertices = new Point[vertices.length];
    for(int index = 0; index < vertices.length; index++) {
        this.vertices[index] = vertices[index];
    }
}

```

Question 6 : Donnez le code des méthodes suivantes :

- `Point getVertex(int index)`
- `void setVertex(int index, Point p)`
- `void translate(int dx, int dy)`
- `int size()`
- `double length()`
- `boolean isClosed()`

```

public class PolygonalChain {
    private final Point[] vertices;

    public Point getVertex(int index){
        return this.vertices[index];
    }

    public void setVertex(int index, Point p){
        this.vertices[index] = p;
    }

    public boolean isClosed(){
        return vertices[0].
            equals(vertices[vertices.length-1]);
    }

    public void translate(int dx, int dy){
        for(int index = 0; index<vertices.length; index++){
            this.setVertex(index,
                getVertex(index).translate(dx,dy));
        }
    }

    public int size(){
        return vertices.length;
    }

    public double length(){
        double sumDistances = 0;
        for(int index = 0; index<vertices.length-1; index++){
            sumDistances += vertices[index]
                .distanceTo(vertices[index+1]);
        }
        return sumDistances;
    }
}

```

Vecteur d'entiers

Question 7 : Donnez le code de la classe Vector.

```
import java.util.Arrays;
```

```
/**
```

```

* La classe Vector implémente un tableau d'entiers
* de taille dynamique. Les éléments du vecteur sont stockés dans un tableau.
* La taille de ce tableau est au minimum doublée à chaque fois qu'il est
* nécessaire de le faire grossir.
*/
public class Vector {

    /**
     * Tableau permettant de stocker les éléments du vecteur.
     * Seuls les size premiers éléments font partie du vecteur.
     * La taille de ce tableau est égale à la capacité du vecteur, c'est-à-dire,
     * au nombre d'éléments maximum que le vecteur peut contenir sans
     * avoir besoin d'allouer un nouveau tableau.
     */
    private int[] elements;

    /**
     * Nombre d'éléments présents dans le vecteur.
     */
    private int size;

    /**
     * Construit un vecteur de capacité initiale initialCapacity.
     *
     * @param initialCapacity Capacité initiale du vecteur
     */
    public Vector(int initialCapacity) {
        this.elements = new int[initialCapacity];
        this.size = 0;
    }

    /**
     * Construit un vecteur de capacité initiale 10.
     */
    public Vector() {
        this(10);
    }

    /**
     * Augmente la capacité du vecteur si nécessaire de façon
     * à permettre le stockage d'au moins minCapacity
     * éléments. S'il est nécessaire d'augmenter la capacité du vecteur,
     * elle est au minimum doublée.
     *
     * @param minCapacity Capacité minimale à assurer
     */
    public void ensureCapacity(int minCapacity) {
        int oldCapacity = elements.length;

```

```

        if (oldCapacity >= minCapacity) return;
        int newCapacity = Math.max(oldCapacity * 2, minCapacity);
        elements = Arrays.copyOf(elements, newCapacity);
    }

    public void resize(int newSize) {
        ensureCapacity(newSize);
        this.size = newSize;
        for (int index = size; index < capacity(); index++){
            elements[index] = 0;
        }
    }

    /**
     * Retourne la capacité du vecteur.
     *
     * @return Capacité du vecteur.
     */
    public int capacity() {
        return elements.length;
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void add(int element) {
        resize(size()+1)
        set(size()-1, element);
    }

    public void set(int index, int element) {
        if (!indexIsInBounds(index))
            return;
        elements[index] = element;
    }

    public int get(int index) {
        if (!indexIsInBounds(index))
            return 0;
        return elements[index];
    }

```

```
private boolean indexIsInBounds(int index){  
    return index >= 0 && index < this.size;  
}  
  
}
```