Exercices #6

Pour plus de commodité dans les tests, nous avons rajouté à **`PointPlan`** une méthode pour initialiser un point avec une abscisse et une ordonnée aléatoire (valeurs décimales à une virgule, entre 0 et 10).
La classe **`Polygone`** est implémentée avec un tableau, ce qui rend malaisé l'ajout de nouveaux points, les tableaux étant de taille fixe.
Vous pouvez trouver les codes sources dans l'archive seance6_debut.tgz en suivant le lien :

https://www.lipn.univ-paris13.fr/~santini/RN3/

Écrire une classe **`TestPolygone`** qui permette de :
Question 1 :
Instancier un polygone nommé **`poly`**.

Correction :

```
public class TestPolygone
{
     public static void main( String [] args)
     {
          Polygone poly = new Polygone();
     }
}
```

Question 2 :
Ajouter huit **`PointPlan`** aléatoires.

Correction :

```
          poly.ajouterRandom( 8);
```

Question 3 :
Insérer un nouveau point de coordonnées *(20,20)* en quatrième position.

Correction :

```
          poly.inserer( new PointPlan( 20, 20), 3) ;
```

Question 4 :
Insérer un nouveau point de coordonnées *(25,25)* en treizième position. Avant même de tester le programme, quel résultat attendez-vous ?

Correction :
L'instruction va provoquer un affichage :
" Erreur dans this.inserer()"
car la position à laquelle on cherche a insérer le **PointPlan** *: la position 13, est supérieur à* **TMAX** *la taille max du tableau.*

```
// poly.inserer( new PointPlan( 25, 25), 12) ;
```

Question 5 :
Insérer un nouveau point de coordonnées *(30,30)* en dixième position.

Correction :

```
poly.inserer( new PointPlan( 30, 30), 9) ;
```

Question 6 :
Ajouter un nouveau point de coordonnées *(40,40)*. Avant même de tester le programme, quel résultat attendez-vous ?

Correction :
L'instruction va provoquer un affichage :
" Erreur dans this.inserer()"
car la position à laquelle on cherche a insérer le **PointPlan** *: la position* **taille***=10, or* **taille** *n'est pas strictement inférieur a* **TMAX** *la taille max du tableau. (***TAILLE***=10 et* **taille***=10)*

```
// poly.ajouter( new PointPlan( 40, 40)) ;
```

Question 7 :
Ajouter un nouveau point de coordonnées *(50,50)*. Avant même de tester le programme, quel résultat attendez-vous ?

Correction :
On obtient la même chose qu'a la question 6 pour les même raison.

```
// poly.ajouter( new PointPlan( 50, 50)) ;
```

Vous devriez obtenir une sortie ressemblant à :

```
poly ; 8 ; (5.9,7.8)(5.4,9.5)(2.3,3.4)(4.8,1.2)(4.4,4.6)(3.2,2.8)(2.1,4.5)(1.3,8.7)
poly ; 9 ;
(5.9,7.8)(5.4,9.5)(2.3,3.4)(20.0,20.0)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)
Erreur dans this.inserer() poly ; 9 ;
(5.9,7.8)(5.4,9.5)(2.3,3.4)(20.0,20.0)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)
poly ; 10 ;
```

```
(5.9,7.8)(5.4,9.5)(2.3,3.4)(20.0,20.0)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(30.
0,30.0) Erreur dans this.ajouter() poly ; 10 ;
(5.9,7.8)(5.4,9.5)(2.3,3.4)(20.0,20.0)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(30.
0,30.0) Erreur dans this.inserer() poly ; 10 ;
(5.9,7.8)(5.4,9.5)(2.3,3.4)(20.0,20.0)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(4.8,1.2)(30.
0,30.0)
```

Exercices - Suite

Nouvelle implémentation de Polygone

Dans l'exercice précédent, vous avez vu quelques limitations des tableaux. Les questions suivantes visent à produire une nouvelle implantation pour **`Polygone`**, de manière à ce que tous les ajouts et insertions de l'exercice précédent soient couronnés de succès (sauf l'ajout en treizième position). Cette nouvelle implantation pourra utiliser les **`ArrayList`** vus en cours.
L'interface publique de la classe **`Polygone`** ne doit pas changer, de manière à assurer une compatibilité avec toutes les classes qui utiliseraient déjà la version *tableaux* de la classe **`Polygone`**. Vous utiliserez donc tel quel le **`TestPolygone`** précedent. Vérifier qu'un code modifié continue de rendre au moins les mêmes services que la version précédente, c'est faire ce que l'on appelle un *test de non régression*.

Question 8 :
Donnez les nouvelles variables d'instance de la classe **`Polygone`**.

Correction :
Plus besoin ni de **`TAILLE`**, *ni de* **`taille`**.

```
public class Polygone
{
     private ArrayList <PointPlan> points;
     private String nom;

     ...
}
```

Question 9 :
Réécrire les deux premiers constructeurs.

Correction :
Le second constructeur ne change pas.

```
     public Polygone () {
          nom="";
          this.points  = new ArrayList<PointPlan>();
     }
     public Polygone (String nom) {
```

```
        this();
        this.nom = nom;
    }
```

Question 10 :
Réécrire tous les accesseurs (getters, setters etc)

Correction :
Les méthodes **`getNom()`** *et* **`setNom()`** *ne changenet pas.*

```
    public PointPlan getPoint (int i) {
        return this.points.get(i);
    }
    public void setPoint(int pos, PointPlan p)  {
        this.points.add(pos, p);
    }
    public void ajouter(PointPlan p) {
                this.points.add(p);
    }
    public int nbPoints() {
        return this.points.size();
    }
    public String getNom(){
        return this.nom;
    }
    public void setNom(String nom)  {
        this.nom =nom;
    }
```

Question 11 :
Réécrire **`insérer(...)`**. **Attention :** ce n'est pas parce qu'un **`ArrayList`** augmente sa taille en fonction des besoins qu'il est possible d'insérer des éléments après une case qui n'existe pas.

Correction :

```
    public void inserer(PointPlan p, int pos) {
        if (pos<this.nbPoints()){
            this.points.add( pos, p);
        }
        else if( pos == this.nbPoints()){
            this.points.add(p);
        }
        else {
            System.out.println("Erreur dans this.inserer()");
        }
    }
```

Question 12 :
Est-ce que **`translater(...)`** a besoin d'être réécrite ? Si oui, faites-le d'une manière qui reste également compatible avec la première version du programme. C'est ainsi que la méthode **`translater(...)`** aurait dû être codée dès le début.

Correction :
Une version compatible avec un tableau statique, comme elle aurait dû être codée dès le début :

```
public void translater (float dx, float dy) {
     for (int i = 0; i < this.nbPoints(); i++) {
          this.getPoint(i).translater(dx,dy);
     }
}
```

Une version propre au **`ArrayList`** :

```
public void translater (float dx, float dy) {
     for (PointPlan p : this.points) {
          p.translater(dx,dy);
     }
}
```

Question 13 :
Même question avec **`creerTranslater(...)`**, **`toString(...)`** et **`ajouterRandom(...)`**.

Correction :
Il n'est pas nécessaire de modifier ces 3 méthodes. Elles restent valides quand on change pour un **`ArrayList`**.

Question 14 :
Réécrire les deux derniers constructeurs. Pour vous aider, la méthode **`Arrays.asList(tab)`** retourne un **`ArrayList`**, transformation du tableau **`tab`** en **`ArrayList`**.

Correction :

```
public Polygone(String nom, PointPlan[] tab){
     points=(ArrayList<PointPlan>)Arrays.asList(tab);
     this.nom = nom;
}
public Polygone (Polygone p)
{
                this( p.nom) ;
     this.nom = p.nom;
     for (PointPlan pt : p.points)
          this.ajouter( new PointPlan(pt));
}
```

Documentation d'une classe

Question 15 :

Réalisez une javadoc complète de la classe **`Polygone`**