Exercices #5

Vous trouverez en téléchargement sur le lien suivant les JavaDoc et les sources des classes **`PointPlan, Polygone, Triangle`** et **`Quadrilatère`**. Les classes ont été augmentées de quelques méthodes pour faciliter la résolution de ce TD.

Question 1 :
Donnez l'entête de la classe **`Polygone`** pour signifier que cette classe réalise l'interface **`Comparable`**.

Correction :

```
public class Polygone implements Comparable
```

Question 2 :
Écrire dans **`Polygone`** la méthode permettant de remplir ce contrat. On considère qu'un **`Polygone`** est inférieur à un autre si son périmètre est plus petit.

Correction :

```
public class Polygone implements Comparable
{
     …
     /**
      * Définit un ordre sur les Polygones
      * @param o l'objet à comparer au Polygone courrant
      * @return –1, 0 ou 1
      */
     public int compareTo(Object o)
     {
          double per1 = this.périmètre() ;
          double per2 = ((Polygone)o).périmètre() ;
          if ( per1 < per2 )
               return -1;
          if ( per1 == per2 )
               return 0;
          return 1;
     }
}
```

Question 3 :
Dans un programme de tests :

- Définissez deux références c1 et c2 à des objets de type **`Comparable`**.
- Faites référencer à la première un nouveau **`Triangle`** avec trois points : *(2001,2000)*, *(2000, 2001)*, *(2001,2001)*
- Faites référencer à la seconde un nouveau **`Quadrilatere`** avec quatre points *(1,1)*, *(1000,1)*, *(1000,1000)*, *(1,1000)*

Correction :

```
public class Test
{
     public static void main( String [] args )
     {
          Polygone c1;
          Polygone c2;

          PointPlan [] t = new PointPlan [3] ;
          t[0] = new PointPlan(2001,2000);
          t[1] = new PointPlan(2000, 2001);
          t[2] = new PointPlan(2001,2001);

          PointPlan [] q = new PointPlan [4] ;
          q[0] = new PointPlan(1,1);
          q[1] = new PointPlan(1000, 1);
          q[2] = new PointPlan(1000,1000);
          q[3] = new PointPlan(1,1000);

          try
          {
               c1 = new Triangle( t, 150);
               c2 = new Quadrilatère( q, 150);
               if (c1.compareTo(c2)==1)
                    System.out.println("Supérieur strict");
               else
                    System.out.println("Inférieur ou égal");
          }
          catch (Exception e)
          {
               System.err.println(e);
          }
     }
}
```

Question 4 :
Qu'affichent les instructions suivantes ?

```
{
     if (c1.compareTo(c2)==1)
          System.out.println("Supérieur strict");
     else
          System.out.println("Inférieur ou égal");
}
```

Correction :

```
Inférieur ou égal
```

Question 5 :
Proposez une implémentation alternative de la méthode **`compareTo()`** pour que les **`Polygone`**s soient comparés selon la distance de leur barycentre à l'origine du repère orthonormé.

Correction :

```
public int compareTo(Object o)
{
     double bary1 = this.barycentre().distanceAOrigine() ;
     double bary2 = ((Polygone)o).barycentre().distanceAOrigine() ;
     if ( bary1 < bary2 )
          return -1;
     if ( bary1 == bary2 )
          return 0;
     return 1;
}
```

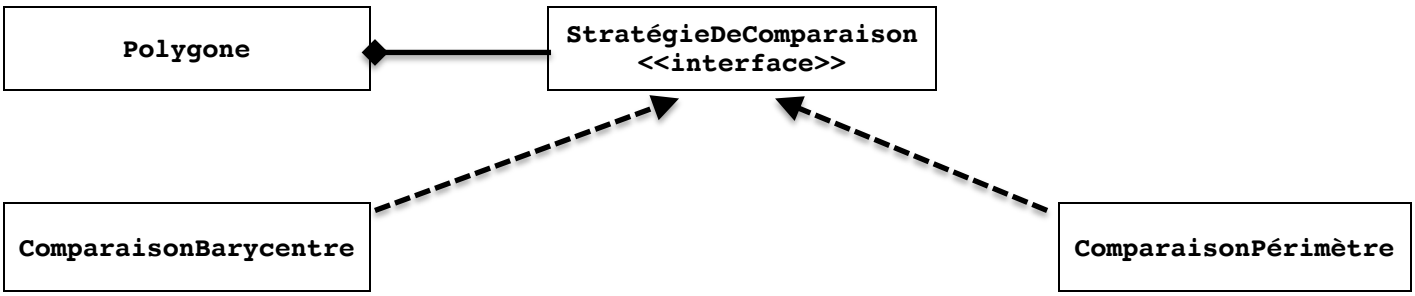
L'affichage obtenu avec le script de test est alors :

```
Supérieur strict
```

Nous nous proposons d'implémenter le patron de conception Strategy pour la comparaison des polygones.
Le patron de conception stratégie est utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application. Le patron stratégie est prévu pour fournir le moyen de définir une famille d'algorithmes, encapsuler chacun d'eux en tant qu'objet, et les rendre interchangeables. Ce patron laisse les algorithmes changer indépendamment des clients qui les emploient.

Exercices

Ici, l'algorithme que nous souhaitons pouvoir changer est l'algorithme utilisé par la méthode **`compareTo()`** ; Nous souhaitons pouvoir passer dynamiquement, pour un jeu de polygone donné, d'une comparaison basée l'ordre défini sur les périmètre à l'ordre définit sur la distance des barycentre à l'origine du repère. La description de ce pattern est la suivante :



Question 6 :
Définissez une nouvelle interface **`StratégieDeComparaison`**. Cette interface doit proposer un contrat simple, avec une seule méthode **`comparaison(Polygone p1, Polygone p2)`**, retournant `-1`, `0` ou `1` de manière similaire à **`compareTo(.)`**

Correction :

```
public interface StratégieDeComparaison
{
      /**
       * Compare deux polygone et renvoie -1 si p1{@literal <}p2,
       * à 0 si p1==p2 et +1 si p1{@literal >}p2.
       * @param p1 le premier polygone de la comaraison
       * @param p2 le second polygone de la comparaison
       * @return -1, 0 ou +1 selon le résultat de la comparaison.
       */
      public int comparaison( Polygone p1, Polygone p2) ;
}
```

Question 7 :
Définissez une nouvelle classe **`ComparaisonBarycentre`** qui réalise l'interface **`StratégieDeComparaison`**. Inspirez-vous largement du code de **`compareTo(.)`** de la Question 5 pour écrire la méthode **`comparaison(...)`** de la nouvelle classe.

Correction :

```
public class ComparaisonBarycentre implements StratégieDeComparaison
{
      /**
       * Compare 2 polygones sur la base de la distance à l'origine
```

```
      * de leurs barycentres
      * @param p1 le premier Polygone de la comparaison
      * @param p2 le deuxième Polygone de la comparaison
      * @return -1, 0 ou +1 selon le résultat de la comparaison
      */
     public int comparaison(Polygone p1, Polygone p2)
     {
          double bary1 = p1.barycentre().distanceAOrigine() ;
          double bary2 = p2.barycentre().distanceAOrigine() ;
          if ( bary1 < bary2 )
               return -1;
          if ( bary1 == bary2 )
               return 0;
          return 1;
     }
}
```

Question 8 :
Définissez une nouvelle classe **`ComparaisonPérimètre`** qui réalise l'interface **`StratégieDeComparaison`**. Inspirez-vous largement du code de **`compareTo(.)`** de la Question 5 pour écrire la méthode **`comparaison(...)`** de la nouvelle classe.

Correction :

```
public class ComparaisonPérimètre implements StratégieDeComparaison
{
     /**
      * Compare 2 polygones sur la base de la distance à l'origine
      * de leurs périmètres
      * @param p1 le premier Polygone de la comparaison
      * @param p2 le deuxième Polygone de la comparaison
      * @return -1, 0 ou +1 selon le résultat de la comparaison
      */
     public int comparaison(Polygone p1, Polygone p2)
     {
          double per1 = p1.périmètre() ;
          double per2 = p2.périmètre() ;
          if ( per1 < per2 )
               return -1;
          if ( per1 == per2 )
               return 0;
          return 1;
     }
}
```

Question 9 :
Définissez dans **`Polygone`** une nouvelle variable de classe publique nommée **`comparateur`**, pour référencer une **`StratégieDeComparaison`**.

Correction :

```
public class Polygone extends Figure implements Comparable
{
     ...

     public static StratégieDeComparaison comparateur;

     ...
}
```

Question 10 :
Réécrivez la méthode **`compareTo(...)`** de polygone, simplement en appelant **`comparaison(...)`** de la variable de classe (statique).

Correction :

```
public class Polygone extends Figure implements Comparable
{
     ...

     /**
      * Définit un ordre sur les Polygones
      * @param o l'objet à comparer au Polygone courrant
      * @return –1, 0 ou 1
      */
     public int compareTo(Object o)
     {
          return this.comparateur.comparaison(this, (Polygone) o);
     }
     ...
}
```

Question 11 :
Proposez un programme testant le changement dynamique de stratégie de comparaison.

Correction :

```
public class Test2
{
     public static void main( String [] args )
     {
          Polygone c1;
          Polygone c2;

          PointPlan [] t = new PointPlan [3] ;
          t[0] = new PointPlan(2001,2000);
          t[1] = new PointPlan(2000, 2001);
          t[2] = new PointPlan(2001,2001);

          PointPlan [] q = new PointPlan [4] ;
          q[0] = new PointPlan(1,1);
          q[1] = new PointPlan(1000, 1);
          q[2] = new PointPlan(1000,1000);
          q[3] = new PointPlan(1,1000);

          try
          {
               c1 = new Triangle( t, 150);
               c2 = new Quadrilatère( q, 150);

               Polygone.comparateur = new ComparaisonBarycentre();;
               if (c1.compareTo(c2)==1)
                    System.out.println("Supérieur strict");
               else
                    System.out.println("Inférieur ou égal");

               Polygone.comparateur = new ComparaisonPérimètre();;
               if (c1.compareTo(c2)==1)
                    System.out.println("Supérieur strict");
               else
                    System.out.println("Inférieur ou égal");
          }
          catch (Exception e)
          {
               System.err.println(e);
          }
     }
}
```