Laboratoire no. 3

Objectifs

- Pratiquer les notions de classe et d'objet
- Pratiquer la représentation UML d'une classe

Exercice 1

Définir une classe `Segment` permettant de modéliser un segment de droite du plan.
Un segment est défini par ses deux points extrémités. Un point extrémité est supposé défini par deux coordonnées entières (abscisse et ordonnée de type *int*).

Cette classe doit permettre :
1. de construire des instances de cette classe
2. de récupérer chacun des points extrémités du segment courant
3. de modifier les coordonnées des points extrémités du segment courant
4. d'afficher, sous la forme `[(x1,y1);(x2,y2)]`, le segment courant
5. de récupérer une représentation du segment courant sous la forme d'un objet de type `String`
6. de récupérer la longueur du segment courant
7. de déterminer si un point donné appartient ou non au segment courant
8. de permuter les deux points extrémités du segment courant
9. d'échanger[1] deux objets de type Segment (à implémenter sous deux formes : une méthode usuelle et une méthode statique)

[1] si le segment s1, respectivement s2, admet A et B, respectivement C et D, comme points extrémités avant l'échange, s1, respectivement s2, aura C et D, respectivement A et B, comme points extrémités après l'échange

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement des diverses méthodes de la classe `Segment`.

NB Pour les commentaires, toujours pas de commentaires "à la Javadoc" pour l'instant ! Idem pour l'exercice 2.

IMPORTANT !
Avant de coder quoi que ce soit, établir à la main (c'est-à-dire sans utiliser un outil logiciel pour ce faire) le schéma UML de la classe `Segment`.
Doivent figurer sur le schéma :
- les champs (attributs) de la classe avec indication de leur type et de leur droit d'accès (en représentation standard)
- les méthodes de la classe avec indication de leur droit d'accès (en représentation standard), de leurs paramètres formels (nom et type) ainsi que du type de leur valeur de retour

Indications :

- Pour le point 5 (classe `Segment`), il s'agit de redéfinir la méthode `toString()` héritée (par toute classe) de la classe `java.lang.Object`
- La classe Segment est supposée ne contenir aucune classe interne
- En ce qui concerne la représentation UML d'une classe, des explications seront données en début de séance de laboratoire

Exercice 2 (partie 1)

On suppose avoir la classe `Point` suivante :

```
public class Point {

   public Point (double x, double y) {
      this.x = x; this.y = y;
   }

   public void déplace (double dx, double dy) {
      x += dx; y += dy;
   }

   public double abscisse() {
      return x;
   }

   public double ordonnée() {
      return y;
   }

   private double x; // abscisse du point
   private double y; // ordonnée du point

}
```

Compléter la classe `Point` ci-dessus en la dotant des méthodes suivantes :

- `homothétie` qui multiplie les coordonnées par une valeur (de type double) fournie en argument
- `rotation` qui effectue une rotation dont l'angle (en radians) est fourni en argument
- `rho` et `theta` qui fournissent les coordonnées polaires du point
- `afficheCartésien` qui affiche les coordonnées cartésiennes du point
- `affichePolaire` qui affiche les coordonnées polaires du point

Ecrire aussi un programme de test qui :

- crée un point A d'abscisse x = 1 et d'ordonnée y = 1
- affiche les coordonnées cartésiennes et polaires du point A
- **déplace** le point A de dx = -1 et dy = -1
- affiche les nouvelles coordonnées cartésiennes et polaires du point A

- crée un point B d'abscisse x = 1 et d'ordonnée y = 1
- affiche les coordonnées cartésiennes et polaires du point B
- réalise une **homothétie** de 2 sur le point B
- affiche les nouvelles coordonnées cartésiennes et polaires du point B

- crée un point C d'abscisse x = 1 et d'ordonnée y = 0
- affiche les coordonnées cartésiennes et polaires du point C
- effectue 8 **rotations** successives de 45 degrés du point C (par rapport à l'origine) et affiche après chaque rotation les nouvelles coordonnées cartésiennes et polaires du point C

IMPORTANT !
Avant de coder quoi que ce soit, établir à la main (c'est-à-dire sans utiliser un outil logiciel pour ce faire) le schéma UML de la classe `Point`.
Doivent figurer sur le schéma :

- les champs (attributs) de la classe avec indication de leur type et de leur droit d'accès (en représentation standard)
- les méthodes de la classe avec indication de leur droit d'accès (en représentation standard), de leurs paramètres formels (nom et type) ainsi que du type de leur valeur de retour

Exercice 2 (partie 2)

Modifier la classe `Point` de l'exercice 2.1 (en faire une nouvelle variante), de telle sorte que les données privées soient maintenant les coordonnées polaires d'un point et non plus ses coordonnées cartésiennes.
Faire en sorte que le "contrat" initial de la classe soit respecté en évitant de modifier les champs publics ou les en-têtes des méthodes publiques (le programme de test de l'exercice 2.1 doit ainsi pouvoir être utilisé sans aucune modification et doit bien sûr fournir des résultats identiques[1])

[1] … aux approximations numériques près !

A réaliser

☐ Seul
☒ Par groupe de 2

Travail à rendre **le 27.10.2015, au début de la séance de laboratoire**

Exercice 1

☒ Fiche de laboratoire (tirage papier)
☒ Schéma UML (tirage papier)
☒ Listings des fichiers sources Java (imprimés avec Notepad++)
☒ Fichiers sources uniquement, dans :
\\eistore1\cours\tic\ RRH\POO1\Rendus\<votre répertoire>\Labo_3\Ex_1
où <votre répertoire> = répertoire du membre du groupe venant en premier dans l'ordre alphab.

Exercice 2

☒ Fiche de laboratoire (tirage papier)
☒ Schémas UML (tirage papier)
☒ Listings des fichiers sources Java (imprimés avec Notepad++)
☒ Fichiers sources uniquement, dans :
\\eistore1\cours\tic\ RRH\POO1\Rendus\<votre répertoire>\Labo_3\Ex_2_1, resp.
\\eistore1\cours\tic\ RRH\POO1\Rendus\<votre répertoire>\Labo_3\Ex_2_2
où <votre répertoire> = répertoire du membre du groupe venant en premier dans l'ordre alphab.