TP JAVA n°3
Interface

Exercice 1 : Bibliothèque

Pour la gestion d'une bibliothèque on nous demande d'écrire une application manipulant des documents de nature diverse : des livres, des dictionnaires, etc.
Tous les documents ont un numéro d'enregistrement et un titre.
Les livres ont, en plus, un auteur et un nombre de pages, les dictionnaires ont une langue et un nombre d'articles. Ces divers objets doivent pouvoir être manipulés de façon homogène en tant que documents.

1) Définissez les classes **Document**, **Livre** et **Dictionnaire**. Donnez à chacune le **constructeur** qui prend autant arguments qu'il y a de variables d'instance à initialiser, et qui se limite à recopier les valeurs des arguments dans les variables correspondantes. Définissez également les **accesseurs** (getNumero(), getTitre(), etc.) nécessaires, ainsi que la méthode **toString()** habituelle.
2) Définissez la classe **ListeDeDocuments** permettant de créer une liste (vide) et d'y ajouter des documents.
3) **void add(Document d)** qui ajoute un document à la liste.
4) **void afficherTousLesDocuments()** qui affiche tous les documents de la liste.
5) **void afficherTousLesLivres()** qui affiche tous les livres (uniquement) de la liste.

Exercice 2 : Point

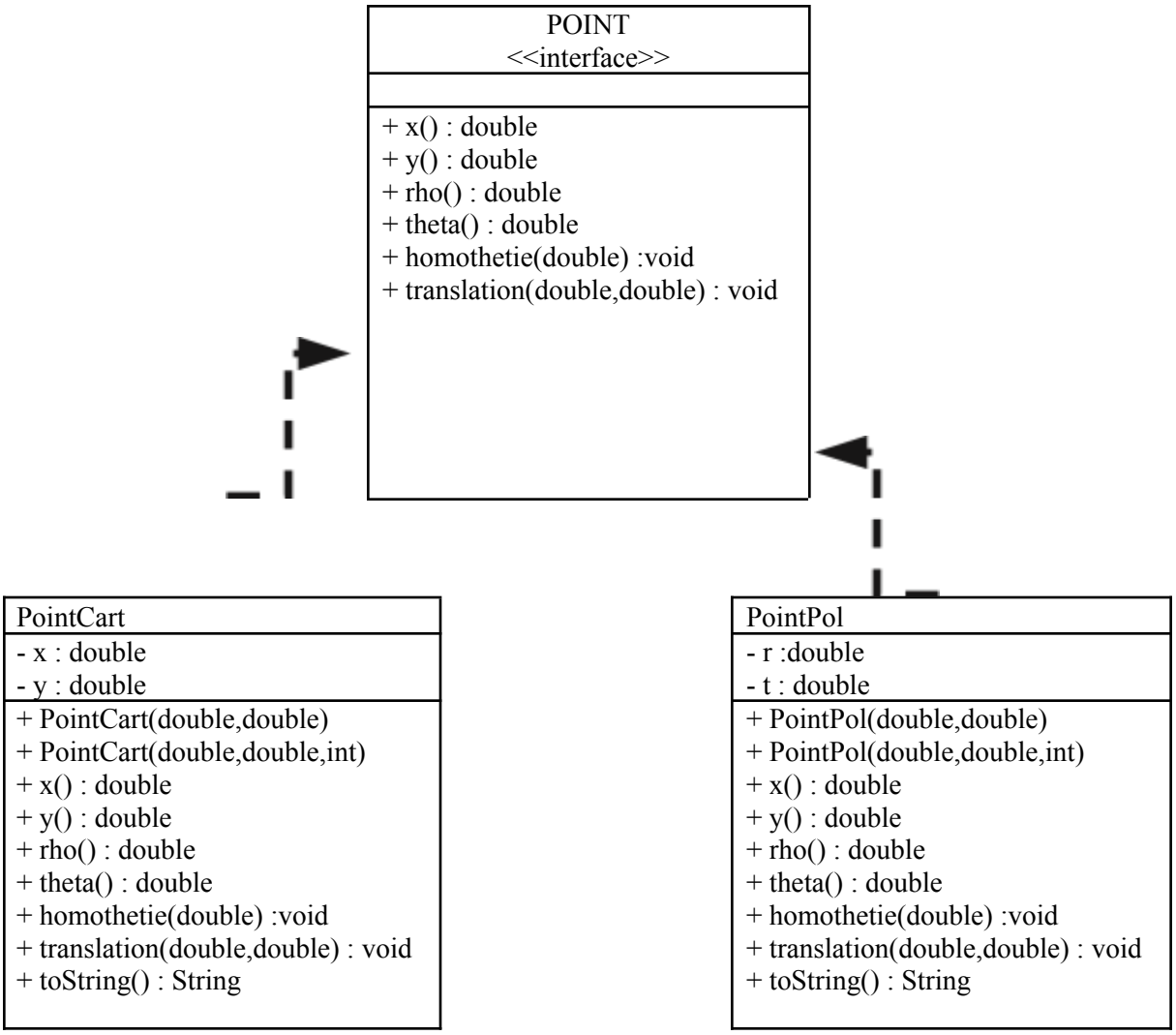
Soient deux classes distinctes :

- **PointCart** (point représenté par ses coordonnées cartésiennes),
- **PointPol** (point représenté par ses coordonnées polaires)

et une interface **Point** rendant les applications utilisatrices indépendantes de l'implémentation choisie dans chaque cas.
Les méthodes de cette interface sont :

```
double x();
double y();
double rho();
double theta();
void homothétie(double k);
void translation(double dx, double dy);
void rotation(double a);
```

Il faudra respecter le diagramme UML suivant :



1) Implementer les classes **PointCart** et **PointPol**. Dans les deux classes on doit avoir les constructeurs pour créer le point voulu quelques soient les données (coordonnées cartésiennes ou coordonnées polaires), il faut donc différencier les constructeurs.
2) Donner le code des fonctions indiquées dans le diagramme UML.
3) Ecrire dans la classe **Main** une fonction permettant à une variable de type Point (paramètre) d'afficher ses coordonnées cartésiennes qu'elle ait été initialisée avec une variable d'un des deux classes.

*Exercice 3 : Hiérarchie de formes géométrique

Dans cet exercice, nous allons essayer de créer une hiérarchie de classes pour représenter différentes formes géométrique qui se base

1) Créez une classe abstraite `Forme` avec les méthodes suivantes :
- `double calculerAire()`
- `double calculerPerimetre()`
- `String getNom()`
- `Double déplacer(Point p)`

2) Implémentez les classes concrètes suivantes qui héritent de `Forme`
    - `Cercle :` Défini par un centre de type Point et un rayon
    - `Rectangle :` Défini par un centre de type Point, une largeur, une hauteur
    - `Triangle :` Défini par trois point

Toutes ces classes doivent implémenter la méthode ToString() qui décrit l'objet (Exemple : Cercle devrait afficher "Cercle de centre (x,y) et de rayon r")

3) Créez une classe GestionnaireFormes qui peut :
    - Stocker une liste de Forme
    - Ajouter et supprimer des formes
    - Calculer la somme des aires de toutes les formes
    - Afficher les détails de toutes les formes
    - Appliquer un déplacement à toutes les formes

Bonus : Ajouter une méthode `bool intersection(Forme f)` à la classe abstraite Forme, et implémenter cette méthode dans toutes les classes dérivées. Cette méthode retourne vrai si deux formes se chevauchent, et faux sinon. Ajouter ensuite une méthode `bool possedeIntersection()` à GestionnaireFormes qui retourne vrai s'il y a deux formes qui se chevauchent dans notre liste de forme.