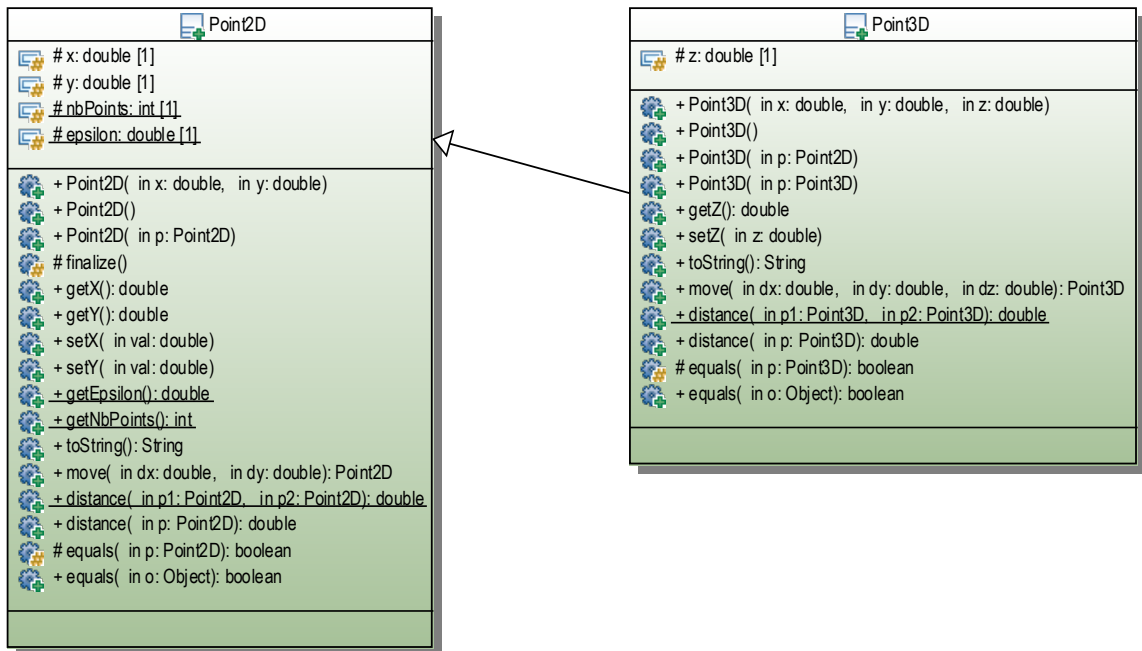


## Introduction

Le but de ce TP est de vous familiariser avec l'environnement de programmation Java, après avoir vu les principes de ce Langage Orienté Objet en cours et en TD. Pour ce faire, nous allons implémenter ce que nous avons vu lors du premier TD.

Vous pourrez trouver une ébauche du code à réaliser dans l'archive : /pub/FISE\_LAOA12/TP1/TP-Figures.zip. Copiez cette archive sur votre compte et dézippez la dans un sous-répertoire (LO par exemple), puis après avoir lancé « eclipse » importez le projet : Import ... → Existing Projects into Workspace → Select root directory → Browse et allez chercher le répertoire dans lequel vous avez dézippé le projet (« TP Figures »). Vous verrez alors apparaître le projet « TP Figures » que vous pourrez alors importer dans Eclipse.

## Travail à réaliser



### 1) Points

- 1) Complétez la classe Point2D du package « points ».
  - a) Les points à compléter dans cette classe sont indiqués par des commentaires commençant par : TODO. La liste des « TODOs » est accessible dans l'onglet « Tasks » : Menu Window → Show View → Tasks ce qui vous permettra de naviguer rapidement entre les « TODOs ».
  - b) Vous pourrez tester votre implémentation en utilisant la classe Point2DTest du package « tests » : Clic Droit → Run As → JUnit Test
- 2) Créez une classe Point3D héritière de la classe Point2D : Clic Droit dans le package « points » → New → Class : Package points, Name Point3D, Superclass points.Point2D.
  - a) Implémentez cette classe comme indiqué dans le diagramme de classes ci-dessus.
  - b) Vous pourrez tester votre implémentation en lançant la classe de test Point3DTest.

## 2) Figures

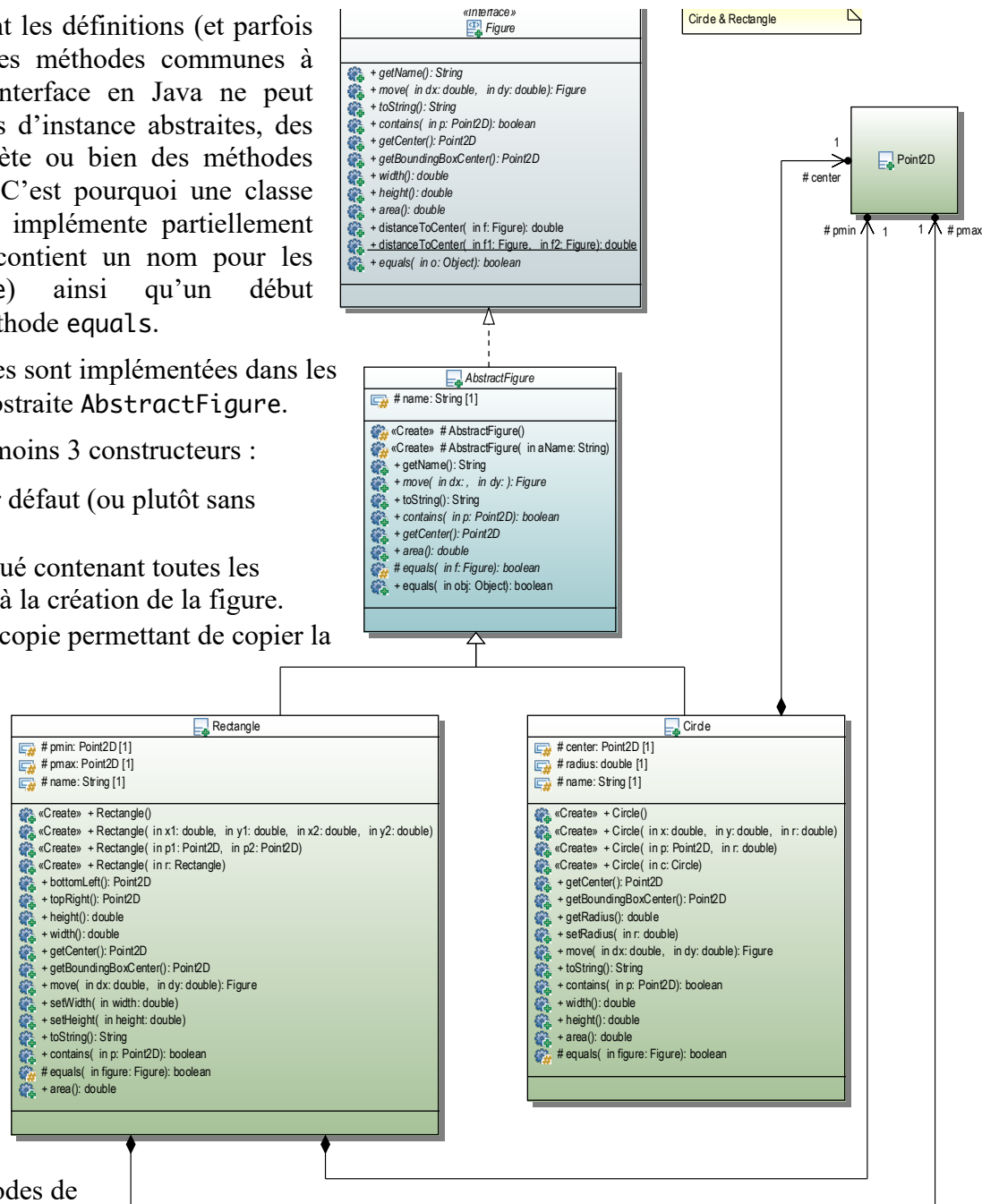
On souhaite implémenter un certain nombre figures géométriques comme des cercles, des rectangles, des polygones, etc. Ainsi que des groupes de figures qui seront alors considérés comme des figures dont la particularité est de contenir d'autres figures.

L'interface Figure contient les définitions (et parfois aussi l'implémentation) des méthodes communes à toutes les figures. Une interface en Java ne peut contenir que des méthodes d'instance abstraites, des méthodes de classe concrète ou bien des méthodes implémentées par défaut. C'est pourquoi une classe abstraite AbstractFigure implémente partiellement l'interface Figure: elle contient un nom pour les figures (String name) ainsi qu'un début d'implémentation de la méthode equals.

Les figures proprement dites sont implémentées dans les classes filles de la classe abstraite AbstractFigure.

Chaque figure possède au moins 3 constructeurs :

- Un constructeur par défaut (ou plutôt sans arguments).
- Un constructeur valué contenant toutes les valeurs nécessaires à la création de la figure.
- Un constructeur de copie permettant de copier la figure.



1) Complétez l'interface Figure ainsi que la classe abstraite AbstractFigure.

2) Complétez la classe Circle héritière d'AbstractFigure. Vous pourrez tester ses constructeurs et ses méthodes spécifiques grâce à la classe de test CircleTest. Vous

pourrez tester les méthodes de l'interface Figure grâce à la

classe de test FigureTest. Le constructeur par défaut crée un cercle de centre (0,0) et de rayon 1. Au fur et à mesure de votre progression dans l'implémentation des différentes figures, vous pourrez décommenter les TODOs dans la classe FigureTest pour inclure de nouvelles figures dans les tests.

3) Créez une classe Rectangle héritière d'AbstractFigure et implémentant l'interface Figure : Clic Droit dans le package « figures » → New → Class : Package figures, Name Rectangle, Superclass figures.AbstractFigure. Et testez là avec les classes de test RectangleTest et FigureTest. Le constructeur par défaut du rectangle construit un rectangle de taille (0,0) centré en (0,0). Notez que les accesseurs en écriture de Rectangle se retrouvent uniquement dans les méthodes : move(double, double), setWidth(double) et setHeight(double).

4) Créez de la même manière que précédemment une classe Polygon héritière d'AbstractFigure implémentant l'interface Figure. La classe Polygon possède une collection de Point2D.

Collection<E> est une interface commune à toutes les collections et qui est implémentée notamment par la classe ArrayList<E> que vous pourrez utiliser pour stocker vos points dans le polygone.

a) Documentation :

- Collection<E> :  
<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Collection.html>
- ArrayList<E> :  
<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/ArrayList.html>

b) Constructeurs :

- Un constructeur sans arguments permettant de créer un polygone par défaut contenant les points suivants :

index	0	1	2	3	4
x	2	4	5	4	2
y	1	1	3	5	5

- Un constructeur avec un nombre variables d'arguments de type Point2D (Polygon(Point2D... pts)). A l'intérieur de ce constructeur l'argument pts peut être considéré comme un tableau et peut donc être parcouru soit comme un tableau, soit comme un itérable (avec une boucle foreach)

```
public Polygon(Point2D ... pts)
{
    ...
    for (int i = 0; i < pts.length;
    i++)
    {
        ...
        points.add(pts[i]);
        ...
    }
    ...
}
```

```
public Polygon(Point2D ... pts)
{
    ...
    for (Point2D p : pts)
    {
        ...
        points.add(p);
        ...
    }
    ...
}
```

- Un constructeur de copie.

c) Bounding Box : la classe Polygon utilise une instance de la classe Rectangle pour décrire sa boîte englobante (méthodes getBoundingBoxCenter(), width() & height()). Il faudra donc veiller à mettre à jour cette boîte englobante à chaque fois que le polygone est modifié (lorsque l'on ajoute ou que l'on retire des points ou lorsqu'il est déplacé).

d) Ajout et retrait de points :

- la méthode add(Point2D p) permettent d'ajouter un point au polygone et de renvoyer true si un point a été ajouté.
  - (a) Si l'on cherche à ajouter un point déjà présent parmi les points du polygone, l'opération doit renvoyer false.
  - (b) Si l'on cherche à ajouter un point null, l'opération add(Point2D p) doit lever une NullPointerException.
- La méthode remove(Point2D p) permet de retirer un point du polygone (si celui-ci fait partie des points du polygone).
- La méthode remove() permet de retirer le dernier point ajouté au polygone (si tant est qu'il ait des points).

- e) La classe Polygon doit implémenter les méthodes abstraites définies dans l'interface Figure et la classe abstraite AbstractFigure.
- Contenu : la méthode contains(Point2D p) renvoie vrai si le point p fait partie des points du polygone ou bien si le point p se situe à l'intérieur du polygone. Vous pourrez trouver l'algorithme sur internet : <http://alienryderflex.com/polygon/>
  - La méthode getCenter() renvoie le centre de masse du polygone alors que la méthode getBoundingBoxCenter() renvoie le centre de sa boîte englobante.
    - Le centre de masse  $(c_x, c_y)$  peut être calculé de la manière suivante :

$$c_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1}) (x_i y_{i+1} - x_{i+1} y_i)$$

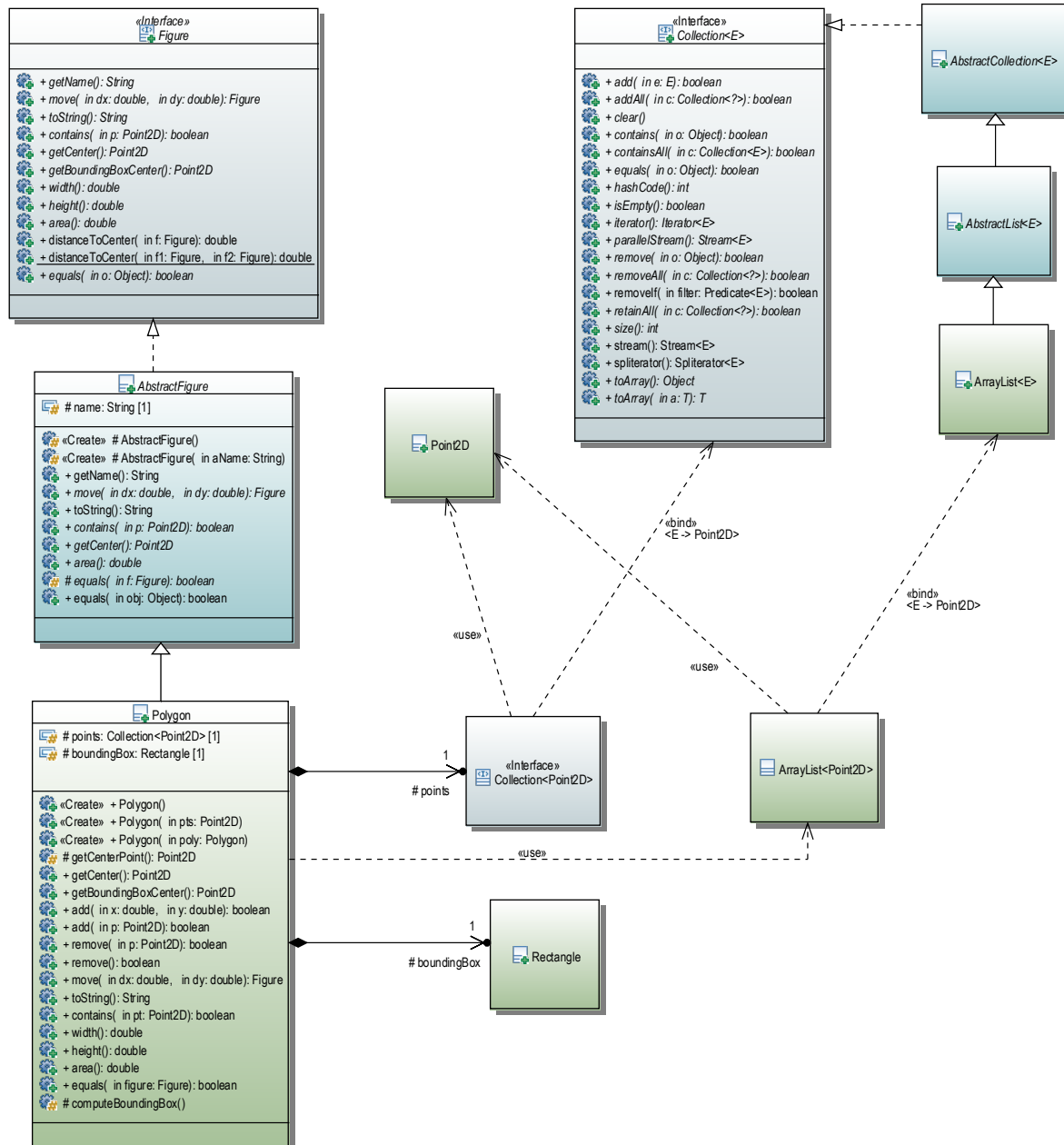
$$c_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1}) (x_i y_{i+1} - x_{i+1} y_i)$$

où A représente l'aire du polygone

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

et  $(x_i, y_i)$  représente les coordonnées du  $i^{\text{ème}}$  point du polygone.

- Le reste des méthodes ainsi que les relations de la classe Polygon sont décrites dans le diagramme de classes suivant.



5) Par héritage de la classe Polygon, créez une classe Triangle.

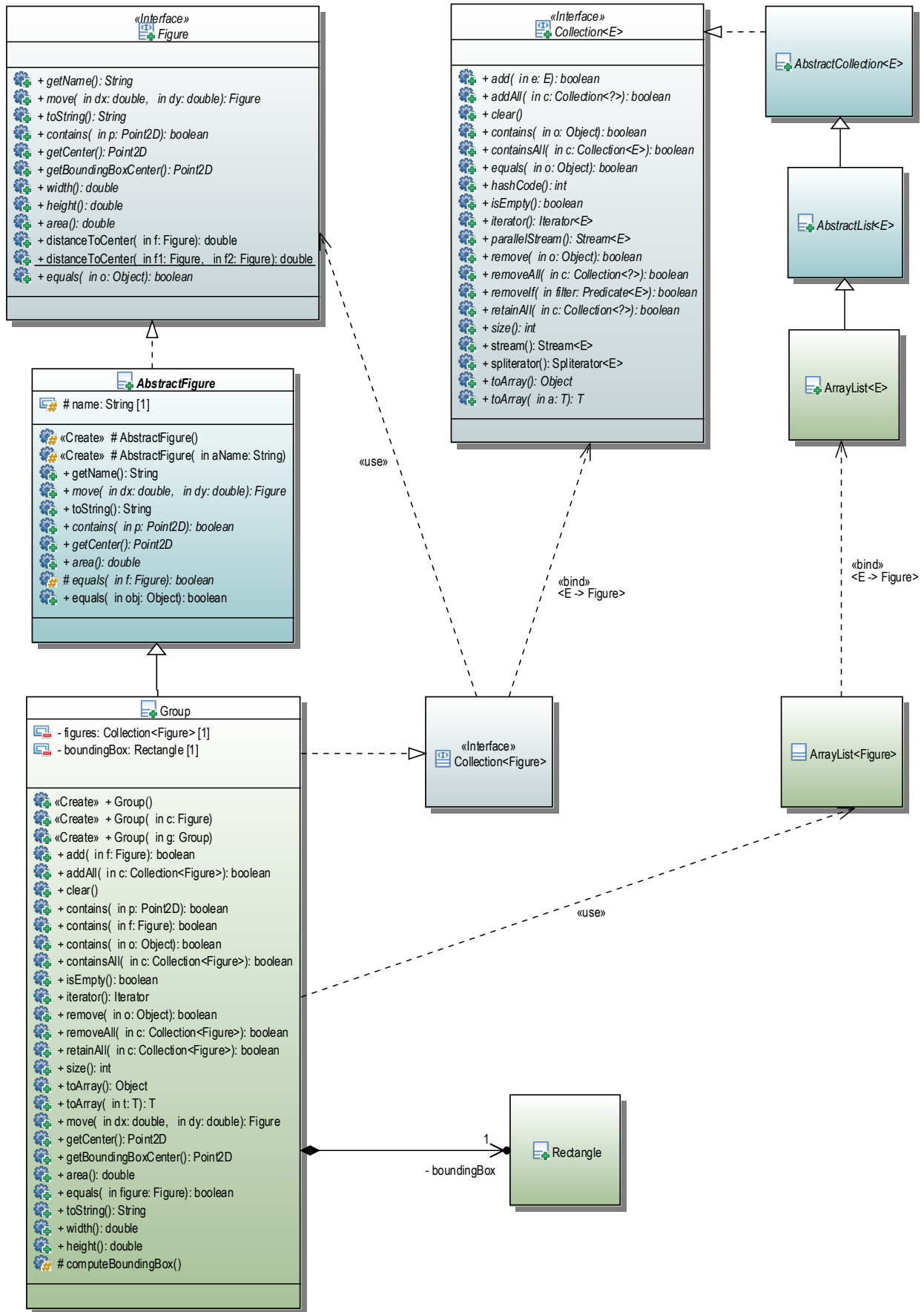
a) Constructeurs : La classe Triangle doit contenir le constructeurs suivants

- Un constructeur par défaut qui crée un triangle contenant les points : [(0, 0), (1, 0), (0.5, 1)].
- Un constructeur valué à partir de trois Point2D.
- Un constructeur de copie

6) On cherche maintenant à faire une classe Group héritant de AbstractFigure et implémentant l'interface Figure qui représente une figure composite (composée d'autres figures). La class Group contient donc une collection de figures (Collection<Figure>) qui lui permet de contenir n'importe quel type de figure (y compris des Group). La classe Group doit aussi à ce titre se comporter comme une Collection<Figure>, elle doit donc aussi implémenter l'interface Collection<Figure> ce qui sera relativement aisé puisqu'elle contient déjà une collection de figures : l'implémentation de cette interface consistera donc (dans la plupart des cas) à faire appel aux méthodes de sa collection interne (une ArrayList<Figure> par exemple) pour implémenter les méthodes définies dans l'interface Collection<E>. Comme la classe Polygon, la classe Group contient un Rectangle permettant de gérer sa boîte englobante qui devra là aussi être mise à jour à chaque fois que le groupe est modifié.

a) Constructeurs. La classe Group doit contenir les constructeurs suivants :

- Un constructeur par défaut qui crée un groupe vide.
  - Un constructeur valué à partir d'une Collection<Figure>.
  - Un constructeur de copie
- b) La classe Group doit implémenter les méthodes abstraites définies dans l'interface Collection<Figure>. Consultez la documentation de l'interface Collection<E> pour déterminer les comportements attendus des méthodes à implémenter.
- On considèrera que l'on ne doit pas ajouter une figure null au groupe, l'opération add(Figure) devra donc lever une NullPointerException si l'on tente d'ajouter une figure nulle. De même on considèrera que l'on ne doit pas ajouter deux fois une même figure dans le groupe. Si l'on tente d'ajouter une figure déjà ajoutée au group l'opération add(Figure) devra renvoyer faux.
  - De même l'opération addAll(Collection<? extends Figure> c) doit lever une NullPointerException si la collection c est nulle ou si elle contient un élément null.
  - On rajoutera une méthode contains(Figure) pour déterminer si une figure particulière fait partie des figures du groupe. On implémentera la méthode générale contains(Object o) de manière à pouvoir tester à la fois la présence d'un Point2D dans au moins une des figures (avec la méthode contains(Point2D) que l'on doit implémenter avec l'interface de Figure) ainsi que la présence d'une Figure parmi les figures du groupe. Par ailleurs, puisque nous avons considéré qu'il ne doit pas y avoir de figure nulle parmi les figures du groupe l'opération contains(Object o) doit lever une NullPointerException si l'objet o est null.
  - De même la méthode containsAll(Collection<?> c) doit lever une NullPointerException si la collection c est nulle, mais on n'exigera pas qu'une telle exception soit levée si la collection c contient un élément null.
  - Cette règle s'applique aussi en principe aux méthodes remove(Object), removeAll(Collection< ?> c) et retainAll(Collection< ?> c) mais nous ne l'exigerons pas ici.
- c) La class Group doit implémenter les méthodes abstraites définies dans l'interface Figure et la classe abstraite AbstractFigure.
- La méthode getCenter() renvoie le centre de masse de l'ensemble des figures du groupe.
  - La méthode getBoundingBoxCenter() renvoie la boite englobante de l'ensemble des figures du groupe. De même les méthodes with() et height() renvoient respectivement la largeur et la hauteur de cette boite englobante.
  - La méthode area() renvoie la somme des aires des figures du groupe.
- d) La classe Group est décrite par le diagramme de classes suivant :



## Annexes

Il est fortement conseillé d'utiliser un environnement de développement intégré comme [Eclipse](#), [NetBeans](#), [IntelliJ](#) ou [Visual Studio Code](#) pour développer efficacement en Java. Néanmoins, il est aussi bon de savoir quels sont les outils sur lesquels se basent ces environnements de développement.

### Java Development Kit

Le Java Development Kit (ou JDK) est l'ensemble des outils nécessaires à la création d'applications Java. Il comprend (parmi d'autres) les outils suivants : un compilateur (javac), une machine virtuelle pour exécuter les objets compilés (java), et un utilitaire pour documenter les classes que vous avez écrites (javadoc).

### Compilation

« **javac** » : le compilateur java.

Ce compilateur génère à partir du fichier XXX.java les différents fichiers XXX.class correspondant à chacune des classes définies dans le fichier XXX.java. On comprendra aisément qu'il est donc judicieux de ne définir qu'une seule classe par fichier afin d'avoir une correspondance \*.class ↔ \*.java.

Pour invoquer la compilation d'un ou plusieurs fichiers .java, on utilise la commande suivante :

```
> javac MaClasse.java [MonAutreClasse.java ...]
```

En principe, la compilation d'un fichier dans lequel on utilise d'autres classes définies dans le même répertoire provoque la compilation de ces autres fichiers. Si tant est que l'on a respecté l'adéquation \*.class ↔ \*.java. Ces fichiers \*.class sont des fichiers contenant du « bytecode » qui correspond au code exécutable par une machine virtuelle Java. Cette machine virtuelle permet de s'affranchir du système d'exploitation propre à telle ou telle machine. N'importe quelle machine virtuelle (quelle que soit la plateforme utilisée) est capable d'exécuter du « bytecode » même compilé sur un autre type de machine.

### Exécution

« **java** » : La machine virtuelle proprement dite.

Pour que la machine virtuelle puisse invoquer l'exécution d'une classe compilée (un fichier bytecode), il faut que cette classe contienne une méthode main définie comme suit :

```
public static void main(String args[]){...}.
```

Pour invoquer l'exécution de la classe contenue dans le fichier compilé « MaClasse.class » on utilisera la ligne de commande suivante :

```
> java MaClasse
```

Les browsers Internet possèdent eux aussi une machine virtuelle qui leur permet d'exécuter des classes java. Néanmoins, ils n'exécutent pas des programmes Java (une classe qui possède une méthode main) mais des Applets (APPLication gadgETS).

### Documentation

« **javadoc** » : utilitaire de documentation.

Javadoc est un utilitaire qui permet de documenter les classes que vous écrivez. La documentation est alors générée au format hypertexte (html) de la même manière que la documentation du JDK. La documentation d'une classe avec javadoc utilise des commentaires particuliers à l'intérieur du fichier \*.java.

```
> javadoc MaClasse.java
```

La documentation du JDK est générée avec ce même outil et est accessible à l'adresse suivante :

<http://docs.oracle.com/javase/21/docs/api/>