Travaux pratiques N°3

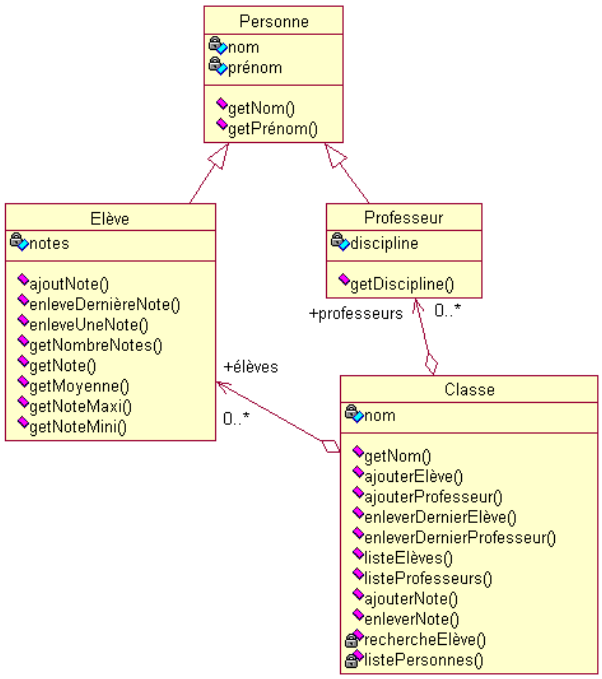
Programmation Orientée Objet en Java

2ème Année Génie Informatique

Exercice 1 : Gestion d'une classe d'élèves

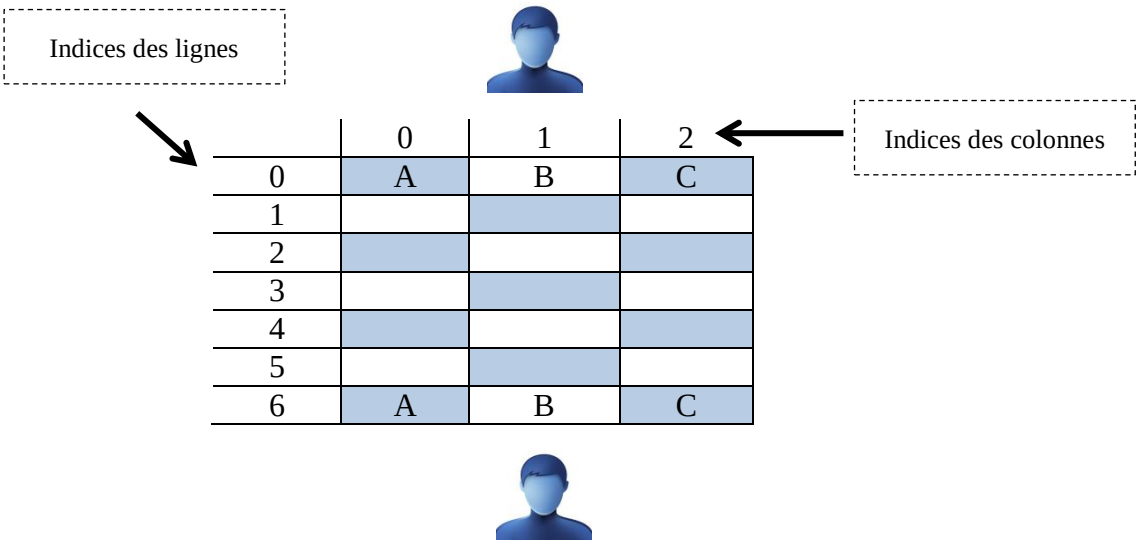
Le digramme ci-dessous représente un diagramme de classes utilisé pour modéliser la gestion d'une classe d'élèves.

1. Expliquer les inconvénients de cette conception ?
2. Proposer une nouvelle conception qui exploite les possibilités du polymorphisme.
3. Implémenter la nouvelle solution en Java.
4. Ecrire un programme de test.



Exercice 2 : Programmation d'un Jeu

On considère un échiquier d'un jeu constitué de 3 pièces A, B et C pour chaque adversaire. L'échiquier et les positions initiales des pièces sont présentés sur la figure ci-dessous.



	0	1	2
0	A	B	C
1			
2			
3			
4			
5			
6	A	B	C

Règles de déplacement des pièces et de calcul des forces pour chaque pièce :

Pièce A : Elle peut être déplacée, que d'une seule case, dans toutes les directions. Sa force égale au carré de l'indice de la ligne où elle se trouve.

Pièce B : Elle peut être déplacée, que d'une seule case, dans toutes les directions, mais elle ne peut pas se déplacer vers des cases grises. Sa force égale 1 si elle se trouve sur une colonne ayant un indice paire et égale 0 sinon.

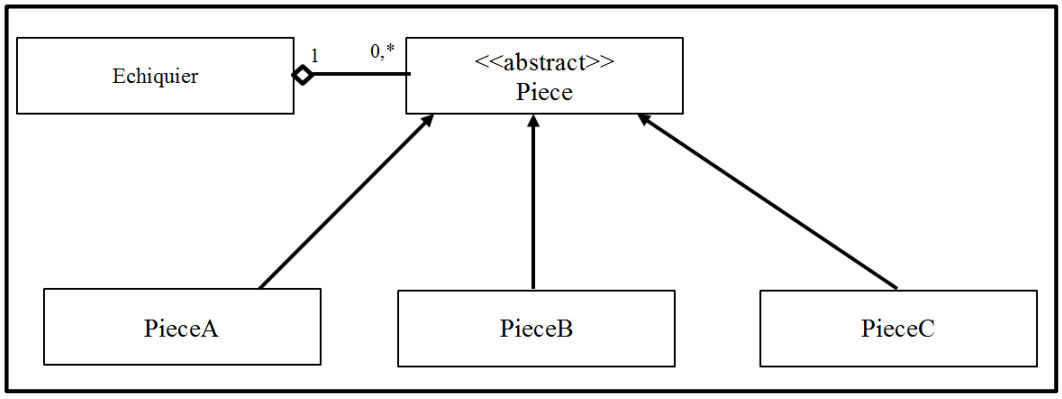
Pièce C :

Elle peut être déplacée, que d'une seule case, dans toutes les directions, mais elle ne peut pas se déplacer vers des cases ayant même indice de ligne et de colonne. Sa force égale la multiplication de l'indice de la colonne et l'indice de la ligne de la case où elle se trouve.

Règles qui s'appliquent à toutes les pièces :

Une pièce peut être déplacée vers une case contenant une pièce de l'adversaire, à condition que sa force soit supérieure ou égale à celle de l'adversaire. Dans ce cas, la pièce de l'adversaire est prise et donc elle doit être éliminée de l'échiquier.

Un échiquier se compose d'une liste de pièces, cf. figure ci-dessous.



La classe **Echiquier** enregistre le score de chaque joueur durant une partie de jeu. Ce score est égal à la somme des forces des pièces prises.

On donne ci-dessous la classe qui représente une pièce de ce jeu.

```
public abstract class Piece {

        protected Echecquier echequier;
        protected Point position;
        protected int couleur;

        public Piece(Point p, int c) {
                position = p;
                couleur = c;
        }

        public abstract List<Point> getPossibleMoves();
        public abstract int getPower();
        public abstract void randomDeplacement();

        …
}
```

1- Écrire la classe **Point** qui représente la position d'une pièce. (Ses coordonnées dans l'échiquier)
2- Ecrire la classe **Echiquier** et sa méthode *removePiece* qui permet d'éliminer une pièce de l'échiquier et de mettre à jour les scores.

3- Écrire les sous-classes concrètes **PieceA**, **PieceB** et **PieceC** qui représentent respectivement les pièces A, B et C. Il faudra donc fournir une implémentation pour les méthodes suivantes :
   - *getPossibleMoves()* : qui donne les mouvements possibles pour une pièces.
   - *getPower* : qui calcule la force d'une pièce.
   - *randomDeplacement* : qui déplace la pièce, en choisissant un déplacement aléatoirement parmi les déplacements possibles.

4- A présent, nous voulons évoluer ce jeu pour qu'il soit possible de sauvegarder les scores des joueurs sur un support de stockage permanant, initialement deux supports sont à envisager : XML et une base de données, cependant votre conception, doit prendre en compte la possibilité d'ajout d'un autre support de stockage, d'une manière souple et sans changement important dans le code.

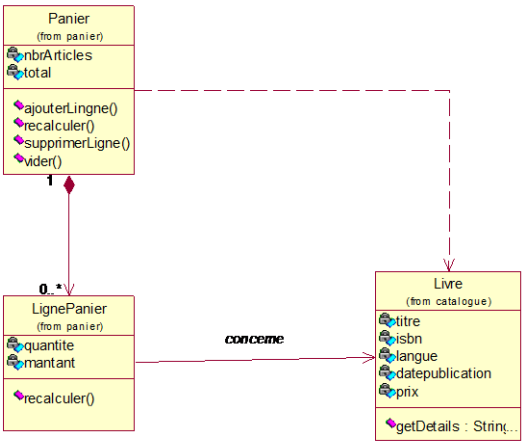
4.1. Donner la conception complète des classes et interfaces Java.

4.2. Ecrire le code nécessaire pour implémenter la partie utilisant une base de données comme support de stockage permanant.

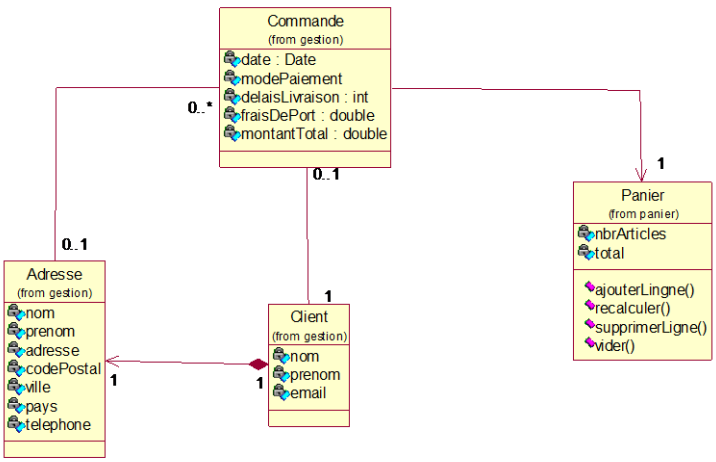
Exercice 3 : Implémentation du modèle métier d'une application E-Commerce

L'analyse métier d'une application E-Commerce spécialisée dans la vente des livres a conduit à l'élaboration du digramme de classe métiers ci-dessous. Implémenter en code java le modèle métier de cette application.

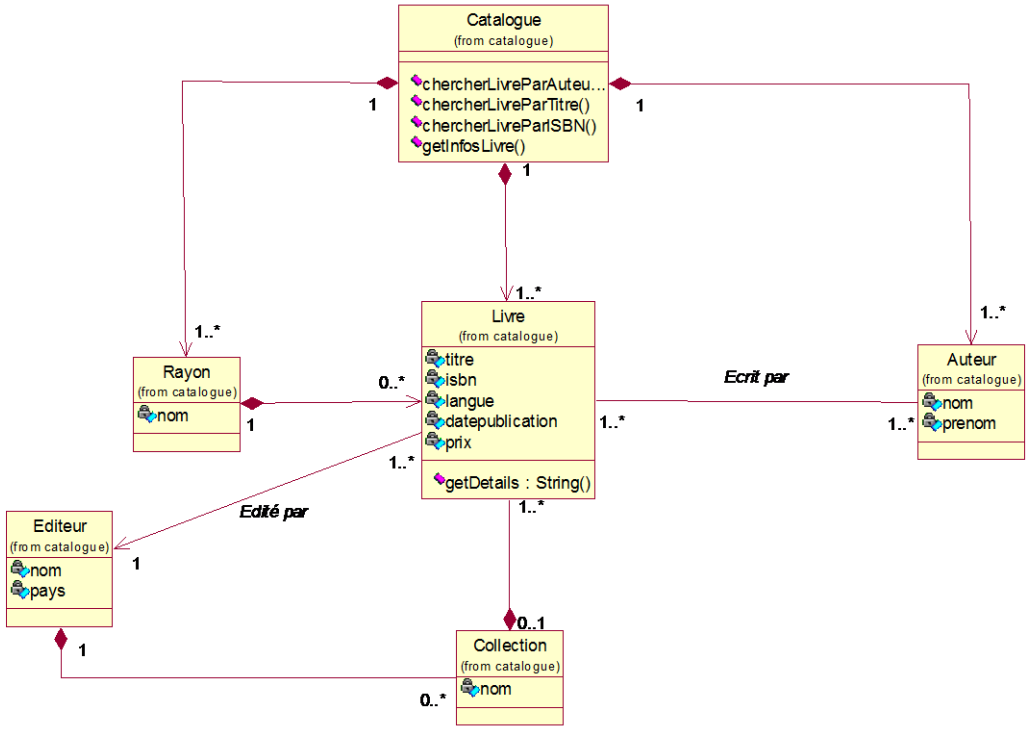
- ***Package : Panier***



- ***Package : Gestion***



- ***Package : Catalogue***



Exercice 4 : Application de dessin des formes tridimensionnelles

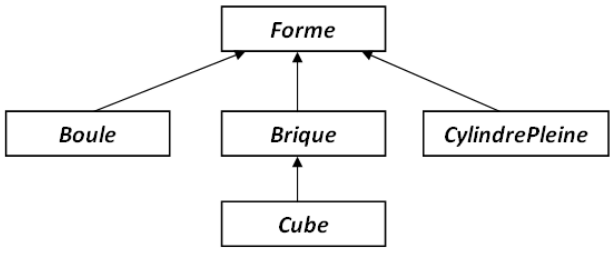
Question 1 :

Ecrire le code de la classe ***Point3D*** qui permet de modéliser un point de l'espace ayant les coordonnées x, y et z.

Doter cette classe des méthodes suivantes :

- Constructeur avec arguments.
- La méthode *equals*
- La méthode *toString*
- Méthodes pour accéder aux différents attributs.
- Méthode *deplacer(double dx, double dy, double dz)* qui permet de déplacer un point dans l'espace. (Exemple *M(x,y,z)* deviendra *M(x+dx, y+dy, z+dz)* après le déplacement) .

- On souhaite disposer d'une hiérarchie de classe permettant de manipuler des formes tridimensionnelles. Pour cela on propose la hiérarchie de classes ci-dessous :



On veut qu'il soit toujours possible d'étendre la hiérarchie en dérivant de nouvelles classes, mais on souhaite pouvoir imposer que ces dernières disposent toujours des méthodes suivantes :

- *calculerSurface :* calculant la surface de la forme.
- *calculerVolume :* calculant le volume de la forme.
- *calculerPoids :* calculant le poids de la forme. (*poids = volume × densité*).

Question 2 : Quelle solution proposez-vous ?

- Les contraintes sur les autres classes sont les suivantes :
  - La classe ***Cube*** ne pourra pas être dérivée.
  - Chaque forme possède un attribut de type ***Point3D*** qui représente son centre de gravité et un attribut réel représentant sa densité. Ces deux attributs sont accessibles via des accesseurs.
    La classe forme dispose d'une méthode *deplacer* permet de déplacer une forme en prenant comme paramètres trois réels représentants les composantes x, y et z d'un vecteur de translation.
  - Un objet de type ***Boule*** est caractérisé par son centre de gravité, sa densité et son rayon.
  - Un objet de type ***CylindrePlein*** est caractérisé par son centre de gravité, sa densité, une hauteur et un rayon.
  - Un objet de type ***Brique*** est caractérisé par son centre de gravité, sa densité, une largeur, une longueur et une hauteur.
  - Un objet de type ***Cube*** est une brique pour laquelle largeur = longueur = hauteur.

De plus, toute forme est capable de donner sa représentation sous la forme d'une chaîne de caractères contenant le nom de sa classe et la description textuelle de chacun de ses attributs.
Exemple : la chaîne de caractères produite pour un objet de la classe Brique :
[Brique centre de gravité : [Point3D x :10.0 , y : 4.0, z : 3.0] densité : 1.2 largeur : 10.5 longueur : 14.3 hauteur : 4.6]
Question 3 : Ecrire le code des classes ***Forme***, ***Brique*** et ***Cube***.
Question 4 : On suppose que la classe ***Boule*** dispose du constructeur suivant :

public Boule(**double** r) qui crée une boule de rayon *r* centrée en l'origine et de densité 1. Etant données les déclarations et initialisations suivantes :

```
Boule b1 = new Boule(100.0); // crée une boule de rayon 100 de centre (0,0,0) et  de densité 1
Boule b2 = new Boule(100.0); //crée une seconde boule de rayon 100 de centre (0,0,0)et de densité 1
```

Quelle est la valeur de l'expression booléenne b1 == b2 ? Justifiez votre réponse.

- L'application permet de créer des documents graphiques, chaque document est constitué de plusieurs formes géométriques. On représente sur les deux diagrammes de classes ci-dessous (Figure 1 et Figure 2) deux approches pour implémenter les classes de domaine de cette application.

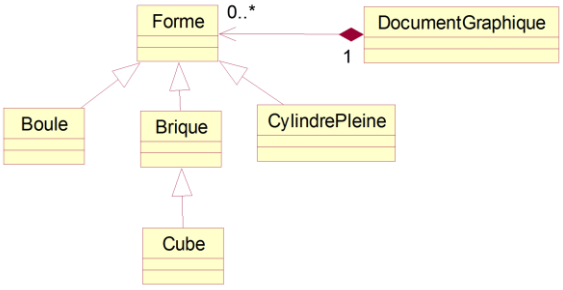


Figure 1

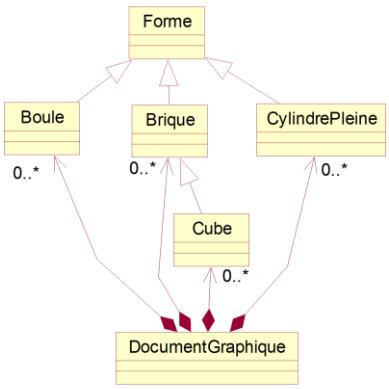


Figure 2

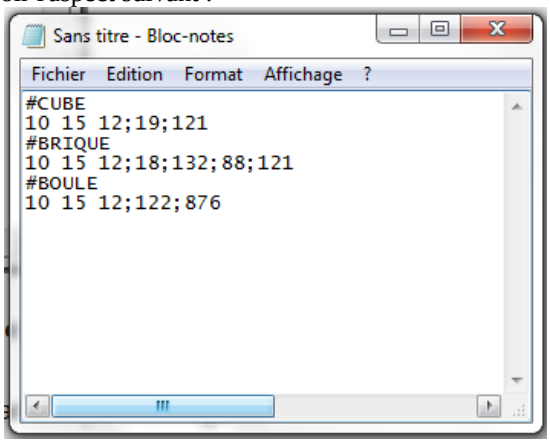
Question 5 : Donner deux inconvénients de l'approche présentée par le diagramme de la *figure 2*.
Question 6 : Expliquer les avantages de l'approche présentée par le diagramme de la *figure 1*.
Question 7 : Quel est le mécanisme de la programmation orientée objet qu'il faut mettre en œuvre pour réaliser la conception présentée sur la *figure 1*.
Question 8 : Implémenter la classe ***DocumentGraphique*** selon l'approche de la *figure 1*. Doter cette classe des méthodes

- *addForme* : Ajoute une forme au document.
- *removeForme(int pos)* : Supprime une forme du document.
- *getAllCubes :* retourne la liste des cubes dessinés sur un document.
- poidsTotal qui permet de calculer le poids total des formes contenues dans le document.

- On suppose à présent qu'on doit doter l'application d'un mécanisme permettant de créer des formes géométrique à partir d'un fichier texte. La structure générale de ce fichier est la suivante :

- Une ligne commençant par le caractère «#» stockant l'intitulé.
- Une ligne stockant les paramètres, séparés par des points virgules

Exemple : Un tel fichier pourra avoir l'aspect suivant :



Question 9 : Définir une exception explicite ***MalformedException***, qui sera levée lorsque l'on tentera de construire une forme ou un point avec des paramètres incorrects. Doter cette exception des constructeurs : MalformedException (String message, Throwable cause) et
MalformedException (String message).

Question 10 :

Créer une classe ***StringUtils*** contenant les méthodes statiques suivantes :

- La méthode ***isACube(String pLine)*** qui retourne ***true*** si la chaîne de caractères passée en paramètre est égale à la chaîne de caractères "#CUBE" et ***false*** sinon.
- La méthode ***extrairePoint(String pLine)*** convertissant une chaîne de caractères de type "x y z" (avec x, y et z sont des réels) en un objet de type ***Point3D***. exemple
  "10 15 12" sera transformée en un objet de type ***Point3D*** ayant les coordonnées x=10, y=15 et z= 12**.** Si la chaîne de caractère passée en argument de la méthode ***extrairePoint*** ne respecte pas le format précédent la méthode lève une exception ***MalformedException.***
- La méthode ***extraireCube(String pLine)*** convertissant une chaîne de caractères de type "x y z ; c ;d" (avec x, y, z, c et d sont des réels) en un ***Cube***. Si la chaîne passée en paramètre ne représente pas un cube la méthode lève l'exception ***MalformedException.***
  Exemples : "10 15 12 ;1;9" sera transformée en un cube de centre (10,15,12) de coté 1 et de densité 19

Question 11 :

On suppose que la classe ***StringUtils*** précédente possède déjà la méthode *readFile* ayant la signature :
public static String[] readFile(String pFileName) throws IOException
permettant de lire le fichier contenant la définition des formes et renvoie un tableau de String contenant **toutes les lignes** de ce fichier.
Utiliser les services de la classe ***StringUtils*** pour implémenter la méthode *initDocWithCubesFromFile(String pFileName)* ;dans la classe ***DocumentGraphique*** ; permettant d'initialiser un document avec la liste des cubes récupérés d'un fichier dont le nom est passé en paramètre. Lors de cette initialisation les lignes incorrectes doivent être ignorées sans arrêter l'opération d'initialisation.

Exercice 5 : Une classe pour gérer les polygones

En géométrie euclidienne, un polygone est une figure géométrique plane, formée d'une suite cyclique de segments consécutifs et délimitant une portion du plan. Le polygone le plus élémentaire est le triangle : un polygone possède au moins trois sommets et trois côtés (cf. Figure 1).

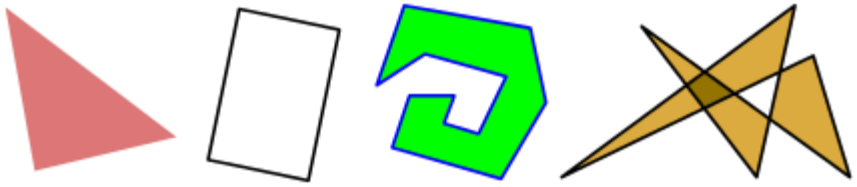


Figure 1 : Exemples de polygones

L'API Java Standard offre une classe ***Polygon*** dans le package *java.awt*, avec cette classe un polygone est représenté par deux tableaux de nombres, un pour les abscisses et un autre pour les ordonnées, ce qui rend son utilisation inconfortable.

On propose dans cet exercice d'écrire une nouvelle classe Polygone où un polygone sera représenté par un tableau d'objets de type Point.

N.B. : Dans cet exercice on donnera une grande importance lors de la correction à la qualité de la conception des classes et à la réutilisation et maintenabilité du code grâce à l'exploitation de l'héritage et polymorphisme.

Questions

1. Écrire une classe **Point** permettant de décrire les coordonnées d'un point dans le plan. Cette classe a :
   - Deux attributs **x** *(l'abscisse)* et **y** *(l'ordonnée)* de type double.
   - Un attribut nom de type caractère représente le nom d'un point.
   - Un constructeur ***public Point(double x, double y, char nom)*** qui permet d'initialiser le point lors de sa création.
   - Des méthodes ***public double getNom(), public double getX()*** et ***public double getY()*** qui retournent respectivement les valeurs de x, de y et de l'attribut nom.
   - Ecrire une méthode ***public String getXmlPresentation()*** qui retourne une chaîne de caractères contenant la représentation XML d'un point, en respectant la syntaxe suivante : Un point ayant le nom='A', x= X0 et y= Y0 on le représente en XML avec la balise <**point nom**="A" **X**="X0" **Y**="Y0"/>.

2. Ecrire une classe **Polygone** représentant un polygone par un tableau d'objet de type Point. Cette classe comportera les méthodes suivantes :
   - ***public Polygone(Point[] sommets)*** : construction d'un polygone à partir du tableau de ses sommets, si le nombre de sommets est inférieur ou égale à 2 il faut lever une exception ***InvalideNombreSommetException*** (**Cette exception de type *Exception* est à définir**).
   - ***public double aire()*** : calcul de la surface du polygone. Sachant que l'aire S d'un polygone ayant les sommets $(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots, (x_{n-1}, y_{n-1})$ est :

$$S = \frac{1}{2} \times |(x_0 + x_1) \times (y_0 - y_1) + (x_1 + x_2) \times (y_1 - y_2) + \cdots + (x_{n-2} + x_{n-1}) \times (y_{n-2} - y_{n-1}) + (x_{n-1} + x_0) \times (y_{n-1} - y_0)|$$

   - ***public Point getSommetMaxX()*** : permet d'obtenir le point d'abscisse maximale du polygone. Au cas où il y en aurait plusieurs, on retournera celui ayant la plus petite ordonnée.
   - Une méthode abstraite ***getXmlPresentation()*** destinée à être implémentée dans les classes filles pour donner la représentation XML de chaque forme géométrique héritant d'un polygone.

3. Définir une classe ***Triangle*** qui représente un triangle, sous-classe de ***Polygone***, avec un constructeur ***public Triangle(Point a, Point b, Point c)*** qui permet de construire un triangle ayant les trois sommets indiqués.
   Donner également une implémentation de la méthode ***getXmlPresentation,*** sachant qu'un triangle ayant les sommets A(X0,Y0), B(X1,Y1) et C(X2,Y2) se présente par le code XML suivant :

   *<**triangle**>*
   *<**point nom**="A" **X**="X0" **Y**="Y0"/>*

```
    <point nom="B"  X="X1"  Y="Y1"/>
    <point nom="C"  X="X2"  Y="Y2"/>
</ triangle>
```

4. Ecrire une classe ***RectangleHorizontal*** qui représente un rectangle horizontal, sachant qu'un rectangle est un polygone, La classe ***RectangleHorizontal*** doit hériter de la classe ***Polygone***. Le constructeur de la classe ***RectangleHorizontal*** doit prendre trois paramètres, le premier est un objet de type Point et représente le coin inférieur gauche du rectangle, le second est de type double et représente la longueur du rectangle, le troisième est de type double et représente la largeur du rectangle.
   La classe ***RectangleHorizontal*** dispose également d'une implémentation de la méthode ***getXmlPresentation*** mais il n'est pas demandé de l'écrire dans cet exercice. Cette méthode pourra être utilisée dans la suite de l'exercice.

5. On suppose à présent qu'on veut réaliser des dessins de constructions (immeubles, bâtiments, maisons, mosquées,…) à l'aide de polygones (cf. figure 2), ainsi on définit la classe ***Construction*** qui se compose d'une collection de polygones.

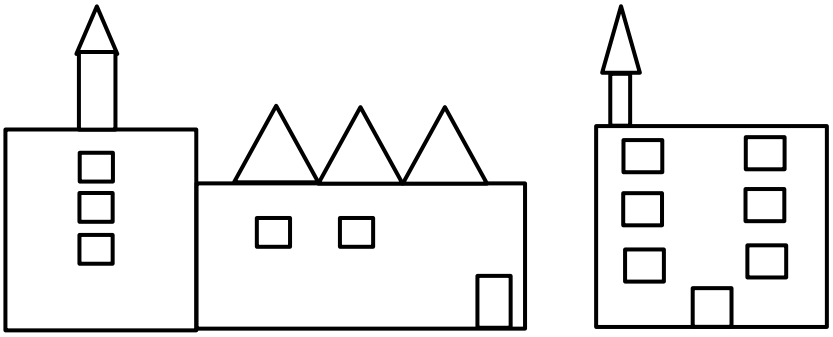


Figure 2 : Exemples des dessins de constructions réalisées à l'aide des polygones

Ecrire la classe ***Construction*** qui dispose des méthodes suivantes :

- Un constructeur permettant d'initialiser la collection des polygones constituant la construction
- ***addPolygone*** : Ajoute un polygone dans la collection des polygones constituant la construction
- ***getXmlPresentation*** permettant de donner une représentation XML d'une construction, en respectant la syntaxe des polygones et en la complétant si nécessaire.

```
<desin>
    ...
    <triangle>
        <point nom="A"  X="X0"  Y="Y0"/>
        <point nom="B"  X="X1"  Y="Y1"/>
        <point nom="C"  X="X2"  Y="Y2"/>
    </ triangle>
    ...
    <rectangleHorizontal>
        <point nom="D"  X="X3"  Y="Y3"/>
        <point nom="E"  X="X4"  Y="Y4"/>
        <point nom="F"  X="X5"  Y="Y5"/>
    </ rectangleHorizontal >
    ...
</desin>
```