

TP Interface

Déroulement :

Vous allez dans ce TP reprendre l'ensemble des notions abordées dans le support : «06_Heritage_Polymorphisme_Interface ».

Pour cela, vous

- construirez une hiérarchie de classes et d'interfaces,
- implémenterez les concepts d'**héritage**, d'**implémentation** et de **composition**.

Vocabulaire utilisé : encapsulation, classe, héritage, classe abstraite, interface, polymorphisme, agrégation, ...

Environnement technique : J2SE, JDK, un EDI (type *Eclipse/NetBeans*).

La documentation **Java Oracle** :

<https://docs.oracle.com/javase/tutorial/java/index.html>

Au cours de ce TP, vous mettrez en évidence :

- La notion de **classe usuelle**.
- La notion de **méthode**.
- La notion d'**héritage**.
- La notion de **classe abstraite**.
- La notion d'**interface**.

Exo 1: la calculatrice

reprendre l'exercice sur la calculatrice (TDProgProceduraleJava.pdf exo 2.6)

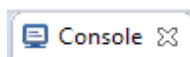
Implémenter des interfaces pour que

- les **additions**
 - retourne le résultat sous forme d'entier (la partie décimale sera tronquée)
- les **soustractions**
 - retourne le résultat sous forme d'entier
- les **multiplications**
 - retourne le résultat sous forme de long
- les **divisions**
 - retourne le résultat sous forme de float

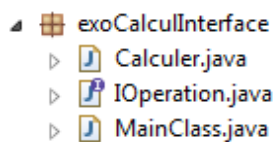
Toutes les opérations se font sur 2 opérandes de type réel

extrait du main:

```
System.out.println("saisissez 2 réels : ");
n1 = sc.nextFloat();
n2 = sc.nextFloat();
if (op == '+') new Calculer(n1, n2).addition();
```



```
operation (+ - * / ) ?
/
saisissez 2 réels :
5
4
je passe par le constructeur explicite de la classe Calculer
division: 1.25
```



Exo 2: figures géométriques 2D

- Figure est un ensemble de figures dessinable potentiellement déplaçable.
- figures présentes dans ce programme : rectangles, triangles, ...
- comportement des objet:
 - rectangles, triangles se dessinent sur la console
 - seul les triangles se déplacent.

Ces figures géométriques se définissent à l'aide de points et de segments

Classe nécessaires pour modéliser l'application:

- une classe **Point** → un point représente une coordonnées x & y.
- une classe **Segment** : un segment a 2 points
- une classe **triangleRect** : un triangle a 3 segments
- une classe **Rectangle** : un rectangle a 4 segments
- une classe mère abstraite **Figure** implémentant les interfaces **IDrawable** & **IMoveable**
- la classe cliente (**MainExoFigure2D.java**) utilisera ces différentes classes

Dans l'interface **IMoveable** la signature de la méthode:

void translate(int dx , int dy);

impose à déplacer les 2 points de chaque segment

Imposer le déplacement de segment demanderait la signature suivante:

void translate(Segment seg, int déplacement);

mais comme un segment est composé de 2 points on peut aussi bien déplacer des points, le concepteur impose le déplacement de point.

force du lien entre composite et composants

Le degré de la relation entre les classes Point/Segment/Figure n'est pas précisé dans le diagramme de classe à vous de le déterminer en implémentant soit agrégation ou composition.

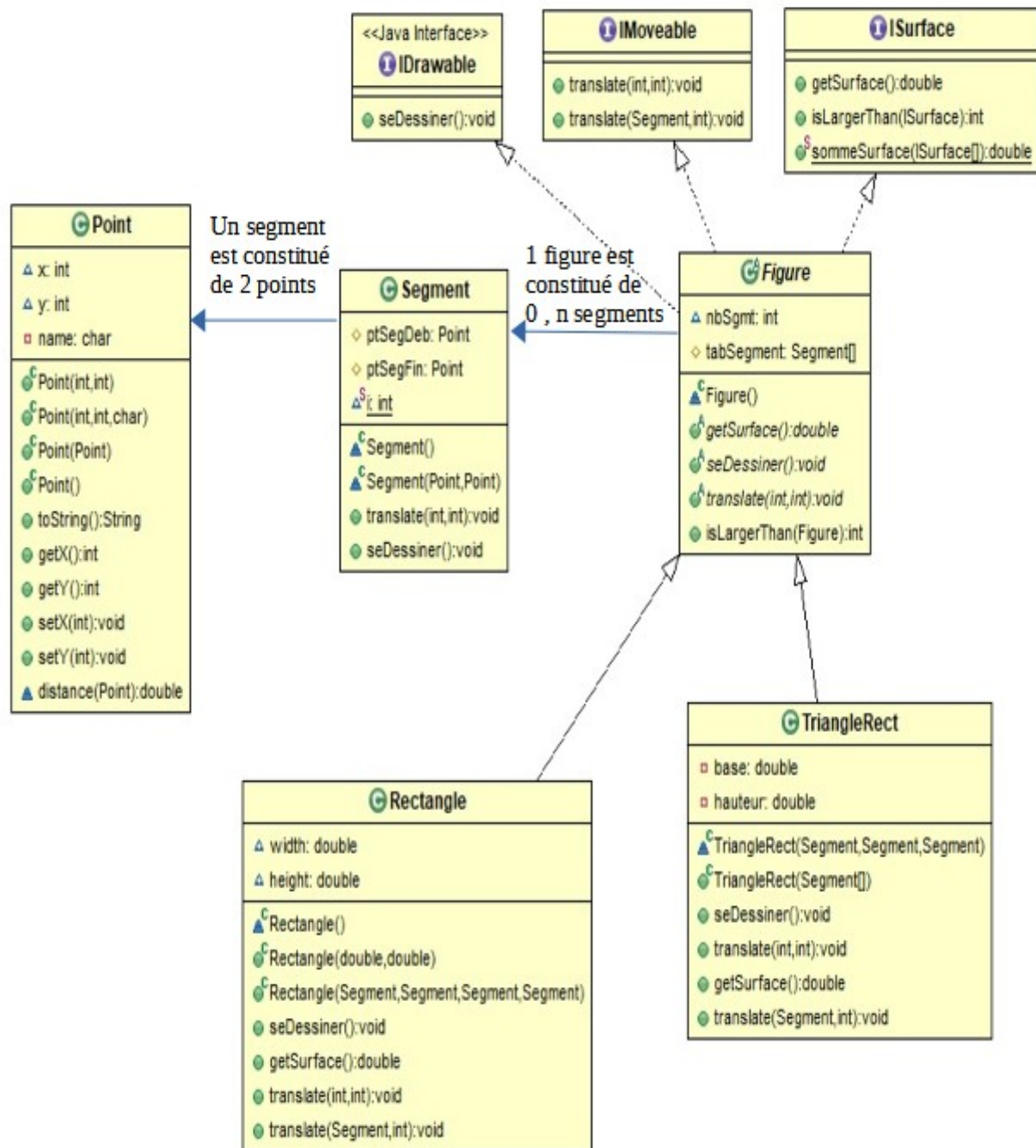


Diagramme de classe : figure 2 D

Relation entre classe : Relation d'héritage, relation d'association

Relation d'héritage : un rectangle est une figure, un triangle rectangle est une figure

Relation d'association entre figure, segment, point

Cardinalité : 0,n 1 figure est constituée de 0 , 3 , 4 segments

Exo 2: figures géométriques (suite)

Écrivez classes et interfaces pour dessiner et déplacer

Ci dessous un exemple d'utilisation des classes figures

```
public static void main( String[] argv ){
    Point ptTriRectEnCePt = new Point( 0 , 0, 'A' );
    Point ptTriSurAbcisse = new Point( 2 , 0, 'B' );
    Point ptTriSurOrdonne = new Point( 0 , 2, 'C' );

    //un triangle ( se construit avec 3 segments donc 3 pts)
    Segment Base = new Segment (ptTriRectEnCePt, ptTriSurAbcisse);
    Segment AjdacentVert = new Segment (ptTriRectEnCePt, ptTriSurOrdonne);
    Segment Hypotenus = new Segment (ptTriSurAbcisse, ptTriSurOrdonne);
    TriangleRect monTriangle = new TriangleRect( Base , AjdacentVert , Hypotenus);

    //un rectangle se construit avec 4 segments donc 4 pts
    //construction du rectangle dans le sens horaire W, X,Y,Z
    Point ptLeftDown = new Point( 0 , 0, 'W' ) ;
    Point ptRightDown = new Point( 0 , 2, 'X' ) ;//height = 2
    Segment HorztDown = new Segment (ptLeftDown, ptRightDown);

    Point ptRightUp = new Point( 3 , 2, 'Y' ) ;
    Point ptLeftUp = new Point( 3 , 0, 'Z' ) ;//width = 3

    Segment HorztUp = new Segment (ptLeftUp, ptRightUp);
    Segment LeftVert = new Segment (ptLeftDown, ptLeftUp);
    Segment RightVert = new Segment (ptRightDown, ptRightUp);
    //un rectangle se construit avec 4 segments
    Rectangle monRect = new Rectangle(HorztDown, RightVert, HorztUp , LeftVert) ;

    Figure figure [] = { monTriangle, monRect } ;
    for( int i = 0 ; i < figure.length ; i++ ){
        figure[i].seDessiner() ; }
    Segment.i= 0;
    System.out.println("deplacement de x= x+1 y= y+1");
    for( int i = 0 ; i < figure.length ; i++ ) figure[i].translate(1, 1); ;
    System.out.println("position des figures apres deplacement" );
    for( int i = 0 ; i < figure.length ; i++ ){
        figure[i].seDessiner() ;
    }
}
```

```
dessin d'un triangle:
segment0->A:0;0 B:2;0
segment1->B:2;0 C:0;2
segment2->C:0;2 A:0;0
dessin d'un rectangle:
segment3->W:0;0 X:0;2
segment4->X:0;2 Y:3;2
segment5->Y:3;2 Z:3;0
segment6->Z:3;0 W:0;0
```

```
deplacement de x= x+1 y= y+1
position apres deplacement
dessin d'un triangle:
segment0->A:1;1 B:3;1
segment1->B:3;1 C:1;3
segment2->C:1;3 A:1;1
position apres deplacement
dessin d'un rectangle:
segment3->W:0;0 X:0;2
segment4->X:0;2 Y:3;2
segment5->Y:3;2 Z:3;0
segment6->Z:3;0 W:0;0
```

Exo 2: figures géométriques (suite)

- la méthode **getSurface** détermine la surface d'une figure, retourne un double

- Déclarer la méthode dans le nouvel interface **ISurface**
- Définissez la méthode dans les classes adéquates
- Sommez les surface de vos figures pour affichage dans la console

```
System.out.println("surface des figures " );  
for( int i = 0 ; i < figure.length ; i++ )  
    System.out.println(figure[i].getSurface()) ;
```

```
surfaces des figures  
Triangle : 2.0  
Rectangle: 6.0
```

Utiliser un Interface comme un Type:

implémenté dans un interface (méthode par défaut) la méthode **isLargerThan**

```
public interface ISurface {  
    public double getSurface();  
    default public int isLargerThan(ISurface other){  
        if (this.getSurface() < other.getSurface()) return -1;  
        else if (this.getSurface() > other.getSurface()) return 1;  
        else return 0;  
    }  
}
```

other est de type interface ISurface

tous les objets sont vues comme étant de type ISurface

- que constatez vous au niveau de la compatibilité ascendante?

la comparaison d'objet hétérogène est alors possible sans avoir besoin de redéfinir la méthode **isLargerThan** dans chaque type (triangle, rectangle...)

par contre la méthode **getArea** doit elle être redéfinie

```
ISurface.java  
ISurface  
    sommeSurface(ISurface[]): double  
    getSurface(): double  
    isLargerThan(ISurface): int
```

manip pretexte pour utiliser un interface comme un type mais comme il existe une classe mère figure vous pouvez aussi bien définir la méthode **isLargerThan** dans cette classe figure:

```
public int isLargerThan(Figure other)
```

INTERFACE

Exo figure (suite)

Dans le main utiliser l'interface ISurface :



```
System.out.println("*****interface ISurface pour comparer 2 figures*****");
System.out.println("*****avec 2 rectangles *****");
Rectangle monRect2 = new Rectangle();//width, height = 0

if ( (monRect).isLargerThan(monRect2) == 0)
    System.out.println("rectangles ont des dimensions identiques");
else if ( (monRect).isLargerThan(monRect2) > 0)
    System.out.println("surface rect1 > surface rect2");
else
    System.out.println("surface rect2 > surface rect1");

monRect2 = monRect;
if ( (monRect).isLargerThan(monRect2) == 0)
    System.out.println(" rectangles ont des dimensions identiques");

System.out.println("*****avec le meme triangle*****");
if ( (monTriangle).isLargerThan(monTriangle) == 0)
    System.out.println("les triangles ont des dimensions identiques");

System.out.println("*****entre rectangle & triangle*****");
if ( (rect1).isLargerThan(monTriangle) > 0)
    System.out.println("surface rect > surface triangle");
if ( (rect1).isLargerThan(monTriangle) < 0)
    System.out.println("surface triangle > surface rect");
```

 Console 

```
*****interface ISurface pour comparer 2 figures*****
*****avec 2 rectangle *****
surface rect1 > surface rect2
les 2 rectangles ont des dimensions identiques
*****avec le meme triangle*****
les 2 triangles ont des dimensions identiques
*****entre rectangle & triangle*****
surface rect > surface triangle
```

Exo figure (fin)

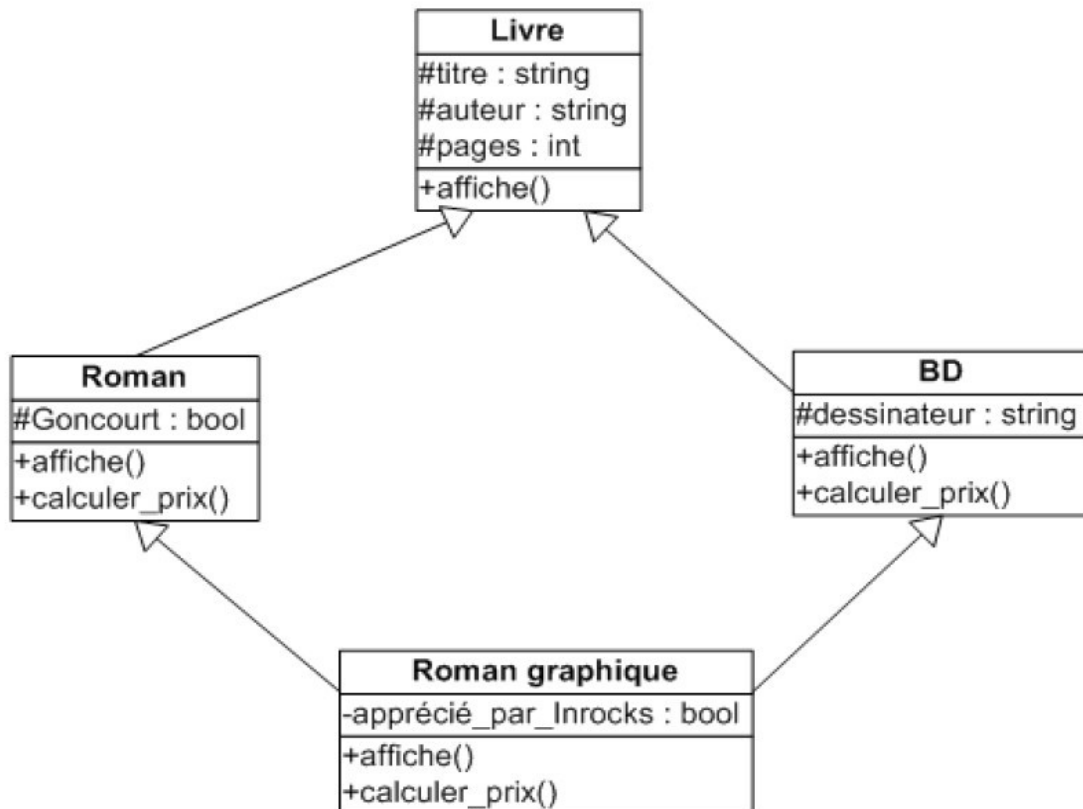
ajouter une méthode sommeSurface calculant la somme de toutes les figures présentes
System.out.println("Somme surf:"+ ISurface.sommeSurface(figure));

extensive work:

ajouter des nouvelles figures géométriques: cercle , carre extends rectangle
remarquez que les ajouts ont peu d'incidences sur le code déjà existant.

correction: exoFigure2D

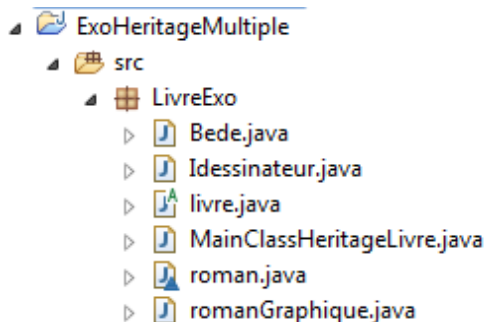
Exo 3 traduire l'héritage multiple du C++ en langage java



```
*****test de la classe roman*****
je passe par le constructeur livre pour titre auteur nombre de page:
je passe par le constructeur de la classe roman pour ajouter BGoncourt
Titre: le meilleur des mondes Auteur: aldous huxley Nombre de pages: 214
prime prix goncourt
prix du roman: 2.14
*****test de la classe bd *****
je passe par le constructeur livre pour titre auteur nombre de page:
je passe par le constructeur de la classe Bd pour ajouter un dessinateur
Titre: Le domaine des dieux Auteur: Goscinny Nombre de pages: 48
nom du dessinateur: Uderzo
prix bd: 14,4
*****test de la classe romangraphique *****
je passe par le constructeur livre pour titre auteur nombre de page:
je passe par le constructeur de la classe roman pour ajouter BGoncourt
Titre: From hell Auteur: Moore Nombre de pages: 500
Pas prime au prix goncourt
et le dessinateur est: Campbell
apprecie inrock
prix du roman graphique: 500.0
```


Exo 3 traduire l'héritage multiple du C++ en langage java

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
  
    System.out.println("*****test de la classe roman*****");  
    roman monRoman = new roman("le meilleur des mondes", " aldous huxley", 214, true);  
    monRoman.affiche();  
    System.out.println("prix du roman: " + monRoman.calculprix());  
  
    System.out.println("*****test de la classe bd *****");  
    classBede mabede = new classBede("Le domaine des dieux", "Goscinnny", 48, "Uderzo" );  
    mabede.affiche();  
    java.text.DecimalFormat df = new java.text.DecimalFormat("0.##");  
    System.out.println("prix bd: " + df.format(mabede.calculprix()));  
  
    System.out.println("*****test de la classe romangraphique *****");  
    romanGraphique monRomanGraphique;  
    monRomanGraphique = new romanGraphique("From hell", "Moore", 500, false, true );  
    monRomanGraphique.setDessinateur("Campbell");//nous ne pouvons pas hérité de la classe Bd  
    monRomanGraphique.affiche();  
    System.out.println("prix du roman graphique: " + monRomanGraphique.calculprix());  
}
```



```
interface IDessinateur  
{  
    void setDessinateur(String name);  
}
```

