UNIVERSITÉ BORDEAUX 1 Sciences Technologies DISVE Licence	**ANNEE UNIVERSITAIRE 2010/2011** **Session 1 de Printemps** **Parcours** : CSB4, MHT4 **UE** : INF252 **Epreuve** : Programmation2 **Date** : 4 Mai 2011 **Heure** : 11h **Durée :** 1h30 **Documents :** Tous documents interdits Le document fait 4 pages. **Epreuve de :** JP.Domenger	Département DL Licence

La clarté, la concision et la précision des réponses seront fortement appréciées par le correcteur. Le barème proposé n'est donné qu'à titre indicatif. Certaines questions sont indépendantes et peuvent être traitées séparément.

Problème.

« Sur les **FigureGeometrique**, *on peut calculer leur surface, leur périmètre, les translater et leur appliquer une homothétie d'un certain rapport et d'un certain centre d'homothétie. Toutes les* **FigureGeométrique** *possèdent un centre de forme qui est un Point2D.*
Les **Polygone** *sont des FormeGeometrique particulières qui sont constituées d'une suite ordonnée de Point2D (les sommet : deux sommets consécutifs forment un segment). Chaque Polygone, peut fournir une copie de la suite ordonnée de ses points (service Point2D donnezSommets()). Les* **Carre** *et les* **Triangle** *sont des Polygones particuliers.*
Il existe également des **Conique** *que l'on peut translater. Toutes les Coniques ne sont pas des FigureGeometrique, mais les* **Cercle** *et les* **Ellipse** *sont des Coniques et aussi des FigureGeometrique»*

Question 1 (1.5 point) : Quelle est la relation qui est associée à l'héritage de type ? Quelles sont les propriétés de l'héritage de type ? Comment se traduit sous forme ensembliste (Union, Intersection, Inclusion) : la relation d'héritage de type (C est un sous type de A), la relation d'héritage multiple de type (C est un sous type de A et C est un sous type de B) ?
La relation d'héritage de type est la traduction de la relation « est un » ou « est une sorte de ».
L'héritage de type a deux propriétés :

1. Le principe de substituions : On peut adresser ou référencer une instance du sous-type en utilisant une variable du sur type.
2. L'interface fonctionnelle du sur type est incluse dans l'interface fonctionne du sous type.

Les instances de C sont compatibles avec les instances de A, l'ensemble C est inclus dans l'ensemble A. C est un sous type de A, donc C est inclus dans A. C est un sous type de B donc C est inclus dans B. Donc C est inclus dans 1'intersection de A et de B.

Question 2 (1,5 point) : Donner le diagramme ensembliste (« diagramme patate ») des spécifications données au début de l'exercice.

On considère une partie de l'interface fonctionnelle publique de l'interface Point2D.

```
public interface Point2D {
```

```
        double getAbscisse() ; // renvoie l'abscisse en coordonnées cartésiennes
        double getrOrdonnee() ;// renvoie l'ordonnée en coordonnées

        public Point2D dupliquerPoint () ;
        public void translation (double dx, double dy) ;
        public void homothetie (Point2D centre, double facteur) ;

        public void positionnerPoint(double abscisse, double ordonnee) ;
        // place le point à la position définie par (abscisse,ordonne) en coordonnées
        cartésiennes.
        ….
}
```

On veut créer une classe de type bibliothèque appelée **GeoLib** qui contient les deux services :

- **Point2D calculCentreForme(Point2D [] tab)** qui calcule le centre de forme du tableau de Point2D tab. Le centre de forme est le point dont les coordonnées sont définies par la moyenne des abscisses et la moyenne des ordonnées des points du tableau tab.
- **Point2D [] creerCopie(Point2D tab[])** qui retourne un tableau qui contient des copies des Point2D contenus dans le tableau tab.

Question 3 (2 points): **Ecrire le code de ces deux services en justifiant s'ils doivent être des services de classe ou des services d'instances.**

La classe Geolib est un module bibliothèque. Les deux fonctions ont pour seul paramètre un tableau de Point, comme la classe tableau de Point n'est pas une classe accessible en Java, les deux services proposés sont donc des services de classe de la classe Géolib.

```
public static Point2D calculCentreForme(Point2D [] tab)  {
int sumX = 0 ; int sumY = 0 ;
for( int i = 0 ; i < tab.length ; i++) {
        sumX += tab[i].getAbscisse();
        sumY += tab[i].getOrdonne();
}
Point2D retour = tab[0].dupliquer();
retour. positionnerPoint(sumX/ tab.length , sumY/ tab.length);
return retour;
}

public static Point2D [] creerCopie(Point2D tab[]) {
Point2D tabRetour = new Point2D[tab.length] ;
for( int i = 0 ; i < tab.length ; i++)
        tabRetour [i] = tab[i].dupliquer();
return tabRetour ;
}
```

On ajoute les spécifications supplémentaires :

- **translater** une FigureGeometrique consiste au moins à translater son centre de forme,
- **translater** un Polygone consiste au moins à translater tous les sommets du polygone,
- faire une **homothétie** de **rapport k** et de **centre centreHomothetie** sur une FigureGeometrique consiste au moins à appliquer sur son centre de forme une **homothétie** de **rapport k** et de centre **centreHomotethie**.
- faire une **homothétie** de **rapport k** et de **centre centreHomothetie** sur un Polygone consiste au moins à appliquer sur tous les sommets du polygone une **homothétie** de **rapport k** et de **centre centreHomotethie**.
- il n'y pas de manière directe de calculer la surface d'un Polygone.

Question 4 (2 points) : Pour chacun des types qui sont en gras dans l'énoncé donnez et justifiez leur déclaration en java (interface, classe, classe abstraite). **On ne demande pas d'écrire le code des services.** Par contre, précisez si le service est redéfini, abstrait, inutile ou factorisé pour les sous classes. Un exemple pour une classe abstraite pourrait être :

```
abstract class Exemple extends C1 {
        public void service1() { code factorisé pour les sous-classes}
        abstract public void service2() ; // service abstrait.
        // public void ser4() ; code inutile le code de C1 convient.
        public ser5() { redéfinition du service ser5() de C1}
}
```

Les questions 4 et 6 sont traitées en même temps.

```
A ajouter dans la classe GeolLib
public static double distance (Point2D p, Point 2D q)
{
        double dx = p.getAbscisse() – q.getAbscisse() ;
        double dy = p.getOrdonnee() – q.getOrdonnee() ;
        return Math.sqrt(dx*dx + dy*dy) ;
}

public abstract class FigureGeometrique {
protected  Point2D centreForme ;
protected FigureGeometrique (Point2D p) { centreForme = p ;}
public abstract double perimetre() ;
public abstract double surface() ;
public Point2D getCentreForme() {return centreForme.dupliquer();}
public void translater (double dx, double dy){ centreForme.translation(dx,dy) ;}
public void homothetie(Point2D centreH, double rapport)
{centreForme.homothetie(centreH, rapport) ;}
}

public abstract class Polygone extends FigureGeometrique {
protected Point 2D [] sommets ;
protected Polygone(Point2D [] tab) {super (Geolib.calculCentreForme(tab);
                                        sommets = tab}
public abstract double surface() ;
```

```
public double perimetre() {
double perim = 0;
for(int i = 1; i< sommets.length; i++)
        perim += Geolib.distance(sommets[i-1], sommets[i]);
perim += Geolib.distance(sommets[sommets.length],sommets[0]);
return perim;
}
public void translater(double dx, double dy) {
super.translater(dx,dy);
for(int i = 0; i< sommets.length; i++)
        sommets[i].translation(dx,dy);
}
public void homothetie(Point2D centreH, double rapport) {
super.homothetie (centreH, rapport);
for(int i = 0; i< sommets.length; i++)
        sommets[i]. homothetie (centreH, rapport);
}
public Point2D donnerSommets(){
        Return Geolib.creerCopie(sommets);
}
}

public class Carre extends Polygone {
double cote;
public carre (Point2D p, double cote){
        super( Carre.creerPolygone(p, cote) ;
}
public double perimetre () {
        return cote*4 ;
}
public double surface (){
        return cote*cote ;
}

public void homothetie (Point2D centreH, double rapport){
super.homothetie(centreH,rapport) ;
double distance = Geolib.distance(centreForme, sommets[0]) ;
cote = distance * 2 / math.sqrt(2) ;
}

private static Point2D [] creerPolygone (Point2D p, double cote){
        Point2D polygone = new Point2D[4];
polygone [0] = p.dupliquer();
polygone[1] = p.dupliquer.positionner(p.getAbscisse()+cote, p.getOrdonnee()) ;
polygone[2] = p.dupliquer.positionner(p.getAbscisse()+cote, p.getOrdonnee()+cote)) ;
polygone[3] = p.dupliquer.positionner(p.getAbscisse(), p.getOrdonnee()+cote)) ;

}
```

```
public class Triangle extends Polygone {
        public Triangle (Point2D [] sommets){ super(sommets);}
        public double surface {// Code à écrire}
        Comme le triangle ne possède aucune spécificité le code de Polygone convient tout à
        fait.
}
public interface conique{
        public void translater (double dx, double dy) ;
}

public class cercle implements Conique extends FigureGeometrique
{
    double rayon ;
    public double perimetre() { à définir}
    public abstract double surface() {à définir}
    public Point2D getCentreForme() ; // inutile même code que FigureGeométrique
    public void translater (double dx, double dy) ; // inutile même code que FigureGeométrique
    public void homothetie(Point2D centreH, double rapport) {
        super.homothetie(centreH, rapport) ;
        plus le specifique pour le rayon.
    }
}
public class ellipse implements Conique extends FigureGeometrique
{
    double grandAxe;
    double petitAxe ;
    public double perimetre() { à définir}
    public abstract double surface() {à définir}
    public Point2D getCentreForme() ; // inutile même code que FigureGeométrique
    public void translater (double dx, double dy) ; // inutile même code que FigureGeométrique
    public void homothetie(Point2D centreH, double rapport) {
        super.homothetie(centreH, rapport) ;
        plus le specifique pour le petit axe et le grand axe.
    }
}
```

Question 5 (1 points). Donner une définition de l'héritage de code ainsi que ses propriétés. Quel est l'intérêt de l'héritage de code. Qu'est ce qu'une classe abstraite ? Quelles sont ses propriétés ? Quel est son intérêt ?

L'héritage de code implique l'héritage de type et donc il implique les deux propriétés de l'héritage de type. Il représente une relation supplémentaire qui est « utilise l'implémentation de ». En effet, les mécanismes de l'héritage de code permettent d'utiliser dans le sous type tout ou partie de l'implémentation du sur type si elle est publique ou protected. La structure d'une instance du sur-type est incluse dans la structure du sur-type.

L'intérêt de l'héritage de code est de proposer un code factorisé utilisable par l'ensemble des sous-types. Attention, le code proposé ne doit pas pénaliser les futurs sous types.

Une classe abstraite est une classe qui fournit une implémentation partielle de son interface fonctionnelle publique ou protected. Comme l'ensemble des services ne possèdent pas de code, elle est non instanciable directement (mais elle le sera par les sous types). Elle permet de proposer un code factorisable pour les sous types qui l'utiliseront. Elle propose à un niveau de la hiérarchie de classe une factorisation de code, sans proposer des implémentations par défauts ou vide des services qui sont abstraits à ce niveau de conception.

Question 6 (3 points): Justifiez et proposez une implémentation pour les classes (abstraite ou non) **FigureGeometrique, Polygone et Carre**.
Voir au dessus.
On dispose maintenant d'un nouveau type qui est le type **Dessinable**, l'interface du type Dessinable est la suivante :

```
public interface Dessinable {
        public void dessiner (GraphicOutput g) ;
}
```

On dispose de plusieurs réalisations du type Dessinable, le service de classe permet aux classes suivantes de mémoriser les segments à dessiner.

```
public class DessinableBord implements Dessinnable
{
        public void stocker(Point2D [] segments){…..};
        public void dessiner (GraphicOutput g) {…..};
                {// ne dessine que les segments définis par le tableau de points}
}

public class DessinableRemplissage implements Dessinnable
{
        public void stocker (Point2D [] segments) {…..};
        public void dessiner (GraphicOutput g) {…..};
                {// dessine les segments et l'intérieur du polygone défini par le tableau
        de points associés à segments}
}
```

Question 7 (1 points) : Ecrire la classe **CarreDessinableBord** qui est un sous type de Carre et une réalisation (implémentation) de **Dessinable**. Quand on dessine un **CarreDessinableBord**, on ne dessine que les segments qui définissent le Carre.

```
public class CarreDessinableBord extends Carre implements Dessinable {
protected DessinnableBord deleg = new DessinnableBord() ;
public CarreDessinableBord (Point2D p, double cote) { super(p,cote) ;}
public void dessiner (GraphicOutput g) {
        deleg.stocker(sommets) ;
        deleg.dessiner(g) ;
}

}
```

Question 8 (2 points) : Ecrire la classe **CarreDessinable** qui est un sous type de Carre et une réalisation (implémentation) de **Dessinable.** Maintenant, le mode de dessin change en fonction de l'état du CarreDessinable. Pour gérer l'état, on ajoute un service supplémentaire dans la classe CarreDessinable qui est :

- **void changerEtat (int typeDessin),** si typeDessin = 0 alors le CarreDessinable se dessine avec seulement ses segments, si typeDessin = 1 alors le CarreDessinable se dessine avec ses segments et son intérieur.

```
public class CarreDessinable extends Carre implements Dessinable {
protected Dessinnable deleg = new DessinnableBord() ;

public CarreDessinable (Point2D p, double cote) { super(p,cote) ;}
public void dessiner (GraphicOutput g) {
        deleg.stocker(sommets) ;
        deleg.dessiner(g) ;
}

public void changerEtat (int typeDessin) {
        if (typeDessin == 0)
                deleg = new DessinnableBord() ;
        if(typeDessin == 1)
                deleg = new DessinnableRemplissage();
}
}
```

Exercice.

Question 9 (0,5 point) Donner une définition du concept de surcharge en Java. Justifiez son intérêt.

La surcharge est la possibilité d'utiliser au sein d'un même contexte (classe), le même nom pour identifier des services qui ont la même sémantique. Pour que la surcharge soit autorisée, il faut les signatures des fonctions diffèrent soit par le nombre de paramètres soit par le type des paramètres à une position donnée. L'intérêt de la surcharge est d'accroitre la lisibilité et la compréhension des classes.

Question 10 (0,5 point) Donner une définition du concept de redéfinition en Java. Justifiez son intérêt.

La redéfinition est la possibilité la même signature de fonction dans un sur-type et un sous – type. Grâce à l'algorithme de sélection des services, l'utilisation de la redéfinition permet de préserver la cohérence du comportement d'une instance indépendamment du type qui le déclare.

Question 11 (2 points) Rappelez les deux étapes de l'algorithme d'exécution des services d'instances en Java.

1. En fonction du type déclaré du sélecteur.
    a. On collecte à partir du type déclaré et dans tous les sur types, l'ensemble des fonctions qui ont le même nom et la même arité.
    b. On supprime toutes les fonctions, pour lesquels la signature de l'appel n'est pas compatible avec la signature de la fonction.
    c. Si il reste plus d'une fonction, on évalue les coûts de conversions entre l'appel et les signatures des fonctions restantes.

d. Si il reste une unique signature, elle est sélectionnée. Si il reste zéro signature, erreur de compilation, si plus d'une erreur de compilation (ambiguité de la surcharge).

2. En fonction du type réel du sélecteur, on remonte dans l'arbre d'héritage jusqu'à trouvé une fonction qui a EXACTEMENT la même signature que celle sélectionnée à l'étape précédente.

Question 11 (3 points) Soient les hiérarchies de classes suivantes :

```
class A {}
class B extends A{}
class C extends B{}

class X {
        public void m(A a){ System.out.println(« X  param A ») ;
}
class Y extends X {
        public void m(C c) {System.out.println(« Y  param C ») ;
        public void m(B b) {System.out.println(« Y  param B ») ;
}

class Z extends Y{
        public void m(B b) {System.out.println(« Z  param  B ») ;
        public void m(A a) {System.out.println(« Z  param  A») ;
}
```

Pour le code ci-dessous, donner pour chacun des 9 appels, les résultats donnés par **chacune des étapes** de l'algorithme d'exécution des services d'instances en Java.

```
public static void main (String [] argv)
{
        C c = new C() ; B b = c ;  A a = c ;
        Z z = new Z() ; Y y = z ;  X x = z ;
        x.m(a) ; // appel 1   y.m(a) ; // appel 2   z.m(a) ; // appel 3
        x.m(b) ; // appel 4   y.m(b) ; // appel 5   z.m(b) ; // appel 6
        x.m(c) ; // appel 7   y.m(c) ; // appel 8   z.m(c) ; // appel 9
}
```

En fonction des appels :

1. Etape 1 m(A) ; Etape 2 : Z param A
2. Etape 1 m(A) ; Etape 2 : Z param A
3. Etape 1 m(A) ; Etape 2 : Z param A
4. Etape 1 m(A) ; Etape 2 : Z param A
5. Etape 1 m(B) ; Etape 2 : Z param B
6. Etape 1 m(B) ; Etape 2 : Z param B
7. Etape 1 m(A) ; Etape 2 : Z param A

8. Etape 1 m(C) ; Etape 2 : Y param C
9. Etape 1 m(C) ; Etape 2 : Y param C

FIN