Chapitre 5

Interface

Les interfaces sont une construction du langage JAVA qui permettent d'abstraire certains comportements des objets de telle sorte que des objets appartenant à des classes différentes puissent être appréhendés sur la base de ce qu'ils ont de semblable.

Prenons un premier exemple. Différents modules d'enseignement du CNAM ont des modalités d'enseignement et d'évaluation différentes. Pour l'enseignement, ils peuvent être en cours du soir, à distance, de jour. Pour l'évaluation, cela peut être par un examen, un contrôle continu, une soutenance. Un examen écrit a une date et laisse une trace écrite permettant une seconde correction en cas de contestation. Il y a une seconde session. Une soutenance orale a une date mais ne permet pas de seconde correction. Un contrôle continu n'a pas nécessairement de date et il n'a pas de seconde session. En dépit de ces différences, il y a un point commun, c'est que chaque méthode d'évaluation débouche sur une note comprise entre 0 et 20, si bien que dans un dossier d'auditeur, on peut faire apparaître les différents types de modules dans un unique relevé de note.

Les différents modules ont vocation à être représentés par des classes différentes parce que certaines de leurs données (variables) et opérations (méthodes) sont différentes, par exemple les dates. Mais ce qu'il y a de commun peut être représenté au moyen d'une interface implémentée par les différentes classes.

Dans ce chapitre, nous allons développer un autre exemple. Il s'agit de représenter des formes géométriques dans le plan. Pour commencer, des cercles, rectangles et triangles. Chaque type de forme a des spécificités. Les données nécessaires pour les représenter sont différentes. Mais ces différentes formes ont en commun d'avoir une surface. La formule de calcul est différente dans chaque classe, mais le résultat est comparable : on peut utiliser la surface pour savoir si un cercle est plus grand qu'un rectangle.

5.1 Les formes géométriques sans interface

Les différentes formes sont définies en utilisant un ou plusieurs points.

```
class Point{
    double x, y;
    Point (double xi, double yi){
        x = xi;
        y = yi;
    }
    static double distance(Point p1, Point p2){
        return Math.sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y) *(p1.y-p2.y));
```

```
    }
}
class Cercle{
    Point centre;
    double rayon;
    Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    double surface(){
        return Math.PI *rayon *rayon;
    }
}
class Rectangle{
    Point basGauche;
    double dimHor, dimVer;
    Rectangle(Point bg, double dh, double dv){
        basGauche = bg;
        dimHor = dh;
        dimVer = dv;
    }
    double surface(){
        return dimHor *dimVer;
    }
}
class Triangle{
    Point p1, p2, p3;
    Triangle(Point p1i, Point p2i, Point p3i){
        p1 = p1i;
        p2 = p2i;
        p3 = p3i;
    }
    double surface(){
        double a = Point.distance(p1,p2);
        double b = Point.distance(p1,p3);
        double c = Point.distance(p2,p3);
        double demiper = (a+b+c)/2;
        return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
    }
}
```

Pour simplifier la définition d'un rectangle, nous avons supposé que les côtés du rectangle sont parallèles aux deux axes du plan.

On voit dans ce programme que les trois classes Cercle, Rectangle et Triangle proposent une méthode pour calculer leur surface. Dans les trois cas, cette méthode ne prend pas de paramètres et renvoie un double. Le code des trois méthodes est différent.

5.2 Les formes géométriques avec une interface

Il est possible de créer une interface pour regrouper ce que ces classes ont en commun. L'interface décrit un ensemble de méthodes. Les classes déclarent *implémenter* l'interface, c'est-à-dire fournir les

méthodes spécifiées. La définition d'une interface crée un type nouveau qui regroupe tous les objets instances de classes qui implémentent l'interface.

```
class Point{ // un point n'a pas de surface
    double x, y;
    Point (double xi, double yi){
        x = xi;
        y = yi;
    }
    static double distance(Point p1, Point p2){
        return Math.sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y) *(p1.y-p2.y));
    }
}
interface AvecSurface{
    double surface(); // Pas de corps de méthode
}
class Cercle implements AvecSurface{
    Point centre;
    double rayon;
    Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    public double surface(){
        return Math.PI *rayon *rayon;
    }
}
class Rectangle implements AvecSurface{
    Point basGauche;
    double dimHor, dimVer;
    Rectangle(Point bg, double dh, double dv){
        basGauche = bg;
        dimHor = dh;
        dimVer = dv;
    }
    public double surface(){
        return dimHor *dimVer;
    }
}
class Triangle implements AvecSurface{
    Point p1, p2, p3;
    Triangle(Point p1i, Point p2i, Point p3i){
        p1 = p1i;
        p2 = p2i;
        p3 = p3i;
    }
    public double surface(){
        double a = Point.distance(p1,p2);
        double b = Point.distance(p1,p3);
        double c = Point.distance(p2,p3);
        double demiper = (a+b+c)/2;
        return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
    }
```

```
}
public class Plan2{
    public static void main(String[] args){
        Point p1 = new Point(1,3);
        Point p2 = new Point(1,5);
        Point p3 = new Point(2,4);
        Triangle t = new Triangle(p1,p2,p3);
        Cercle c = new Cercle(p1,2.5);
        Rectangle r = new Rectangle(p1,2.7,5.0);
        AvecSurface as = t;
        AvecSurface[] tab = new AvecSurface[3];
        tab[0] = new Triangle(p1,p2,p3);
        tab[1] = new Rectangle(p1,2.7,5.0);
        tab[2] = new Cercle(p1,2.5);
        for (int i=0; i<3; i++){
            Terminal.ecrireStringln("surface␣de␣tab[" + i + "]:␣" +
                                    tab[i].surface());
        }
        if ((tab[0].surface() > tab[1].surface())&&
            (tab[0].surface() > tab[2].surface())){
            Terminal.ecrireStringln("tab[1]␣est␣le␣plus␣grand");
        }else{
            Terminal.ecrireStringln("tab[2]␣est␣le␣plus␣grand");
        }
    }
}
```

Quelques règles à suivre :

- l'interface contient la déclaration d'une ou plusieurs méthodes, c'est à dire le type des paramètres et du résultat, le nom et éventuellement des exceptions levées par la méthode. Elle ne contient jamais le code d'une méthode, son corps.
- les classes qui implémentent l'interface doivent le déclarer explicitement après le nom de la classe. Par exemple `class Triangle implements AvecSurface`.
- les méthodes déclarées dans l'interface doivent être définies dans les classes qui l'implémentent avec le même nom, le même type pour les paramètres et le résultat.
- De plus ces méthodes doivent être définies avec le mot-clé `public`. Ce mot-clé ne doit pas apparaître dans la déclaration de la méthode dans l'interface.
- Une variable déclarée avec pour type le nom d'une interface peut contenir une instance de n'importe quelle classe qui implémente l'interface.
- Sur une variable de ce type, on peut appeler toutes les méthodes de l'interface, mais rien que ces méthodes.

Dans notre exemple, les seules opérations disponibles pour les variables `as, tab[0], tab[1]` et `tab[2]` sont l'affectation et l'appel de la méthode `surface`.

5.3 Les formes géométriques avec deux interfaces

A notre programme, nous allons ajouter une nouvelle opération qui est la *translation*, opération qui permet de déplacer une figure sur le plan. Cette opération existe non seulement pour les trois types de formes, mais aussi pour les points. Nous allons creer une nouvelle interface pour regrouper tous les objets qui ont cette opération de translation.

```
interface AvecTranslation{
    void translation(double deplHor, double deplVer);
}
class Point implements AvecTranslation{
    double x, y;
    Point (double xi, double yi){
        x = xi;
        y = yi;
    }
    static double distance(Point p1, Point p2){
        return Math.sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y) *(p1.y-p2.y));
    }
    public void translation(double deplHor, double deplVer){
        x = x + deplHor;
        y = y + deplHor;
    }
}
interface AvecSurface{
    double surface(); // Pas de corps de méthode
}
// Cercle implémente les deux interfaces
class Cercle implements AvecSurface, AvecTranslation{
    Point centre;
    double rayon;
    Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    public double surface(){
        return Math.PI *rayon *rayon;
    }
    public void translation(double deplHor, double deplVer){
        centre.translation(deplHor,deplVer);
    }
}
class Rectangle implements AvecSurface, AvecTranslation{
    Point basGauche;
    double dimHor, dimVer;
    Rectangle(Point bg, double dh, double dv){
        basGauche = bg;
        dimHor = dh;
        dimVer = dv;
    }
    public double surface(){
        return dimHor *dimVer;
    }
    public void translation(double deplHor, double deplVer){
        basGauche.translation(deplHor,deplVer);
    }
}
class Triangle implements AvecSurface, AvecTranslation{
    Point p1, p2, p3;
```

```
    Triangle(Point p1i, Point p2i, Point p3i){
        p1 = p1i;
        p2 = p2i;
        p3 = p3i;
    }
    public double surface(){
        double a = Point.distance(p1,p2);
        double b = Point.distance(p1,p3);
        double c = Point.distance(p2,p3);
        double demiper = (a+b+c)/2;
        return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
    }
    public void translation(double deplHor, double deplVer){
        p1.translation(deplHor,deplVer);
        p2.translation(deplHor,deplVer);
        p3.translation(deplHor,deplVer);
    }
}
public class Plan3{
    public static void main(String[] args){
        Point p1 = new Point(1,3);
        Point p2 = new Point(1,5);
        Point p3 = new Point(2,4);
        Triangle t = new Triangle(p1,p2,p3);
        Cercle c = new Cercle(p1,2.5);
        Rectangle r = new Rectangle(p1,2.7,5.0);
        AvecSurface as = t;
        AvecSurface[] tab = new AvecSurface[3];
        tab[0] = new Triangle(p1,p2,p3);
        tab[1] = new Rectangle(p1,2.7,5.0);
        tab[2] = new Cercle(p1,2.5);
        for (int i=0; i<3; i++){
            Terminal.ecrireStringln("surface␣de␣tab[" + i + "]:␣" +
                                    tab[i].surface());
        }
        AvecTranslation[] tab2 = new AvecTranslation[4];
        tab2[0] = new Triangle(p1,p2,p3);
        tab2[1] = new Rectangle(p1,2.7,5.0);
        tab2[2] = new Cercle(p1,2.5);
        tab2[3] = new Point(1,5);
        // on déplace tout vers la droite
        for (int i=0; i<4; i++){
            tab2[i].translation(2.0,0);
        }
    }
}
```

5.4 Histoire de types

Avec les interfaces, un objet possède plusieurs types java. Le programmeur va choisir l'un ou l'autre de ces types en fonction de ses besoins, mais il devra parfois passer d'un type à un autre.

Par exemple, un objet instance de la classe `Cercle` va pouvoir être placé dans une variable d'un des trois types `Cercle, AvecSurface` et `AvecTranslation`. Dans une variable de type `Cercle`, il aura les deux méthodes `surface` et `translation`. Avec le type `AvecSurface`, il n'aura que la méthode `surface` et avec le type `AvecTranslation`, il n'aura que la méthode `translation`.

Il est possible de faire une affectation d'un objet de type `Cercle` dans une variable de type `AvecSurface` parce que la classe `Cerlce` implémente l'interface `AvecSurface`. Dans l'autre sens, l'affectation n'est pas possible sans conversion de type explicite, car les objets de type `Cercle` ont des caractéristiques que n'ont pas tous les objets du type `AvecSurface`. Par exemple, ils ont les deux variables `centre` et `rayon` que n'ont pas nécessairement les objets du type `AvecSurface`.

```
public class Conv{
    public static void main(String[] args){
        Point p = new Point(1,3);
        Cercle c1 = new Cercle(p,2.3);
        AvecSurface c2 = new Cercle(p,5);
        AvecTranslation c3 = new Cercle(p,4.2);
        c2 = c1; // OK
        c1 = c2; // erreur à la compilation
        c2 = c3; // erreur à la compilation
    }
}
```

Ce programme provoque deux erreurs :

```
> javac Conv.java
Conv.java:78: incompatible types
found   : AvecSurface
required: Cercle
        c1 = c2; // erreur à la compilation
             ^
Conv.java:80: incompatible types
found   : AvecTranslation
required: AvecSurface
        c2 = c3; // erreur à la compilation
             ^
2 errors
```

Il est possible de demander une conversion de type explicite, avec le risque que celle-ci échoue non pas à la compilation mais lors de l'exécution du programme.

```
public class Conv2{
    public static void main(String[] args){
        Point p = new Point(1,3);
        Cercle c1 = new Cercle(p,2.3);
        AvecSurface c2 = new Cercle(p,5);
        AvecTranslation c3 = new Cercle(p,4.2);
        Rectangle r = new Rectangle(p,2,3);
        c2 = c1; // OK
        c1 = (Cercle) c2; // OK
        r = (Rectangle) c2; // compile mais erreur à l'exécution
```

```
    }
}
```

```
> javac Conv2.java
> java Conv2
Exception in thread "main" java.lang.ClassCastException: Cercle
             cannot be cast to Rectangle
        at Conv2.main(Conv2.java:81)
```

On a le droit demander la conversion explicite d'une interface vers n'importe quelle classe qui l'implémente. Plus précisément, une telle conversion de type ne provoque pas d'erreur de compilation. A l'exécution, il y a une erreur si la classe n'est pas effectivement celle de l'objet. On peut constater que la conversion de type ne change strictement rien à l'objet. C'est juste un moyen de changer de type parmi les types qu'a l'objet.

5.5 Programmation abstraite grâce aux interfaces

Supposons que nous n'ayons pas d'interface et que nous voulions comparer la surface de deux figures. Il faudrait dans chaque classe plusieurs méthodes pour comparer un objet de la classe avec un autre objet qui peut être de la même classe ou d'un autre genre de figure. Le code de ces différentes méthodes serait le même, mais le type du paramètre serait différent. Voyons le cas d'une méthode `compareSurface` qui renvoie un entier négatif si l'objet `this` a une surface plus petite, 0 si les deux surfaces sont égales et un entier positif sinon.

Cela donnerait le code suivant pour la classe `Cercle`.

```
class Cercle{
    Point centre;
    double rayon;
    Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    double surface(){
        return Math.PI *rayon *rayon;
    }
    int compareSurface(Cercle c){
        int res=0;
        if (this.surface() < c.surface()){
            res = -1;
        }else if (this.surface() > c.surface()){
            res = 1;
        }
        return res;
    }
    int compareSurface(Rectangle c){
        int res=0;
        if (this.surface() < c.surface()){
            res = -1;
        }else if (this.surface() > c.surface()){
            res = 1;
```

```
        }
        return res;
    }
    int compareSurface(Triangle c){
        int res=0;
        if (this.surface() < c.surface()){
            res = -1;
        }else if (this.surface() > c.surface()){
            res = 1;
        }
        return res;
    }
}
```

Grâce au type `AvecSurface`, on peut écrire une seule méthode qui accepte en paramètre aussi bien un cercle qu'un rectangle ou un triangle.

```
class Point{ // un point n'a pas de surface
    double x, y;
    Point (double xi, double yi){
        x = xi;
        y = yi;
    }
    static double distance(Point p1, Point p2){
        return Math.sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y) *(p1.y-p2.y));
    }
}
interface AvecSurface{
    double surface(); // Pas de corps de méthode
    int compareSurface(AvecSurface as);
}
class Cercle implements AvecSurface{
    Point centre;
    double rayon;
    Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    public double surface(){
        return Math.PI *rayon *rayon;
    }
    public int compareSurface(AvecSurface as){
        int res=0;
        if (this.surface() < as.surface()){
            res = -1;
        }else if (this.surface() > as.surface()){
            res = 1;
        }
        return res;
    }
}
class Rectangle implements AvecSurface{
    Point basGauche;
```

```
    double dimHor, dimVer;
    Rectangle(Point bg, double dh, double dv){
        basGauche = bg;
        dimHor = dh;
        dimVer = dv;
    }
    public double surface(){
        return dimHor *dimVer;
    }
    public int compareSurface(AvecSurface as){
        int res=0;
        if (this.surface() < as.surface()){
            res = -1;
        }else if (this.surface() > as.surface()){
            res = 1;
        }
        return res;
    }
}
class Triangle implements AvecSurface{
    Point p1, p2, p3;
    Triangle(Point p1i, Point p2i, Point p3i){
        p1 = p1i;
        p2 = p2i;
        p3 = p3i;
    }
    public double surface(){
        double a = Point.distance(p1,p2);
        double b = Point.distance(p1,p3);
        double c = Point.distance(p2,p3);
        double demiper = (a+b+c)/2;
        return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
    }
    public int compareSurface(AvecSurface as){
        int res=0;
        if (this.surface() < as.surface()){
            res = -1;
        }else if (this.surface() > as.surface()){
            res = 1;
        }
        return res;
    }
}
public class Plan5{
    public static void main(String[] args){
        Point p1 = new Point(1,3);
        Point p2 = new Point(1,5);
        Point p3 = new Point(2,4);
        Triangle t = new Triangle(p1,p2,p3);
        Cercle c = new Cercle(p1,2.5);
        Rectangle r = new Rectangle(p1,2.7,5.0);
        AvecSurface as = t;
        AvecSurface[] tab = new AvecSurface[3];
```

```
        tab[0] = new Triangle(p1,p2,p3);
        tab[1] = new Rectangle(p1,2.7,5.0);
        tab[2] = new Cercle(p1,2.5);
        for (int i=0; i<3; i++){
            Terminal.ecrireStringln("surface␣de␣tab[" + i + "]:␣" +
                                    tab[i].surface());
        }
        AvecSurface plusGrand = tab[0];
        for (int i=1; i<3; i++){
            if (plusGrand.compareSurface(tab[i])<0){
                plusGrand = tab[i];
            }
        }
    }
}
```