Introduction to Java

Special Care Set for Java Begineers/ Ex 09

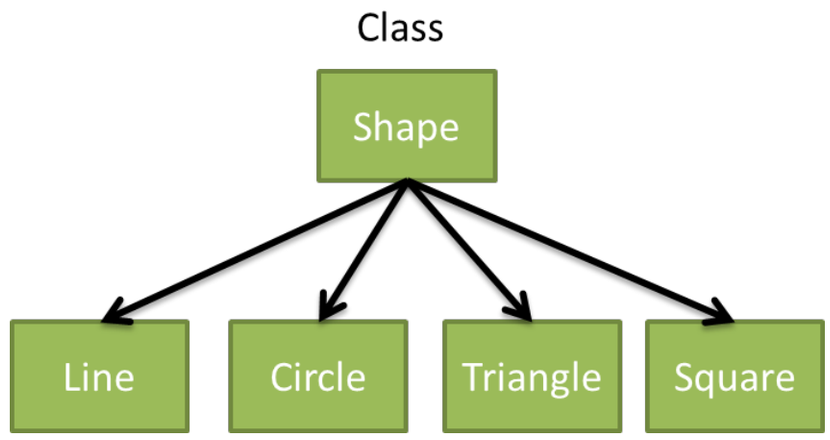
・Notice!
The motivation of this page is to lead regular exercises.
We prepare many useful information in this page as hints of exercises, but this page has no answer.
If you have question, please don't hesitate to ask and discuss your teacher and teaching assistants.
このページは、通常の演習問題への導きとして作られています。
多くの有用な情報をヒントとして準備しておりますが、ここに回答が隠されているとかはありません。
もし、演習に対して質問等ございましたら、遠慮なく教員・TAに質問・議論をしてください。

Example Problem: インターフェース - この機能を実装せよ！ -
Interface - Imprement the function! -

今まで、クラスの作成、クラスの継承による再利用、抽象的なクラス/メソッドによる『作りかけのクラス』の許容など、クラスの作り方を色々と見てきました。
こういったクラスの外側の定義をインターフェースと言いますが、フィールドも持たずに、ただ抽象的なメソッドのみを持っているような『インターフェースのみ』の定義の仕方もJavaには存在します。

今回は、こういった『この機能を実装してください』という『インターフェース』の定義の仕方、使い方を勉強していきましょう。

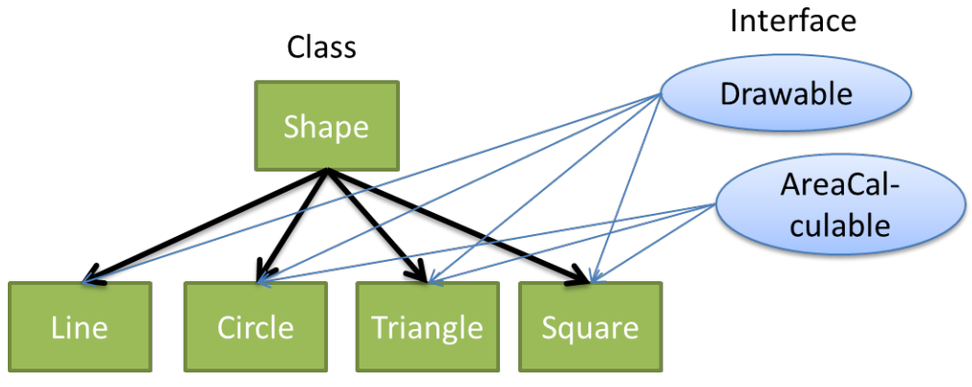
1. クラスや抽象クラスの『継承』と『インターフェース』の違い



Polymorphismの回で、通常のクラスから継承して新たなクラスを作成することや、実装出来ない（若しくはしない）メソッドを『とりあえず』定義することで実体化は出来ないけれどもクラスとして定義できる『抽象』クラスについて勉強して来ました。

『継承』では必ず上位の“1つのクラス”の機能を再利用することになります。右の上図のように、『Shape』からLine, Circle, Triangle, Square等を継承する場合は、Shapeクラス内の public または protected となっているフィールドやメソッドが下位のクラスでそのまま使えるようになり、下位クラスで再定義された場合には、staticメソッドはは『隠蔽（Hiding）』、インスタンスメソッドは『オーバーライド（Override）』となる事を授業で学びました。

また、『Shape』中では実装できないメソッド draw() があった場合、抽象クラスとしてShapeは定義され、各下位のクラスでそれらは実装する必要がありました。



Inteface は、右の下図のようなケースで発生します。 Shape内にdraw()メソッドがあろうがなかろうが、LineやOval等の図形は『描画』出来る必要があるとします。その場合の機能として、draw() メソッドを実装するように指示するものとして "Drawable" インターフェースを追加させます。また、CircleやTriangle, Square等は面積が計算できますから、その面積を計算させるものとして、"AreaCalcurable"インターフェースを追加することもできます。

こういったものは場合によってさまざまに発生します。IS-A関係となるような継承による機能の追加だけでなく、『HAS-A（包含）』関係による機能の追加にも、Interfaceは非常に有効に役立ちます（例: 例外処理 = Throuable, 複数並列処理（スレッド） = Runnableなど）。

また、Interfaceの特徴は、右図のように、1つのクラスに『複数』実装させることが出来ます。IS-A関係が1つだけでは無い場合に、Interfaceはその補完として役に立ちます。

それでは、右下図のような構造のソースコード例を見てみましょう。下の例は、Shape, Line, Circleクラスまでを書いたものです。参考にして、 TriangleクラスやSquareクラスも実装してみましょう。

```
interface Drawable{
        public void draw();
}

interface AreaCalcurable{
        public double areaCalc();
}

public class InterfaceTest {

        public static void main(String[] args) {
                Shape[] shape = {new Line(10,10,100,100), new Circle(100,100,50)};
                print(shape[0]);
```

```
                print(shape[1]);
        }
        static void print(Shape s){
                System.out.println(s.toString());
                s.draw();
        }
}

abstract class Shape implements Drawable{
        protected int x, y, width, height;
        public Shape(){
                x = 0; y = 0;
                width = 0; height = 0;
        }

        public Shape(int x, int y, int width, int height){
                this.x = x; this.y = y;
                this.width = width; this.height = height;
        }

        public String toString(){
                return "x=" + x + ", y=" + y + ", width=" + width + ", height=" + height;
        }

        public void setX(int x){ this.x = x; }
        public void setY(int y){ this.y = y; }
        public void setWidth(int width){ this.width = width; }
        public void setHeight(int height){ this.height = height; }
        public int getX(){ return x; }
        public int getY(){ return y; }
        public int getWidth(){ return width; }
        public int getHeight(){ return height; }
}

class Line extends Shape implements Drawable{
        int x1, y1, x2, y2;

        public Line(){
                super();
        }

        public Line(int x1, int y1, int x2, int y2){
                super(x1, y1, x2-x1, y2-y1);
                this.x1 = x1; this.y1 = y1;
                this.x2 = x2; this.y2 = y2;
        }

        @Override
        public void draw() {
                // TODO Auto-generated method stub
                System.out.println("Draw Line from (" + x1 + "," + y1 +") to (" + x2 + "," + y2 +")");
        }
}

class Circle extends Shape implements Drawable, AreaCalcurable{
        int x1, y1, r;

        public Circle(){
                super();
        }

        public Circle(int x1, int y1, int r){
                super(x1-r, y1-r, r*2, r*2);
                this.x1 = x1; this.y1 = y1; this.r = r;
        }
        @Override
        public double areaCalc() {
                // TODO Auto-generated method stub
                return r*r*Math.PI;
        }

        @Override
        public void draw() {
                // TODO Auto-generated method stub
                System.out.println("Draw Circle: Center (" + x1 + "," + y1 +") with Radius= " + r + " and Area=" + areaCalc());

        }
}
```

出力結果は次の通りです。

```
x: 0, y: 0, Width: 10, Height: 10
x: 10, y: 10, Width: 10, Height: 15
 (x1,y1)=(15,10), (x2,y2)=(20,20), (x3,y3)=(15,25), (x4,y4)=(10,15)
[Rectangle] x: 20, y: 20, Width: 15, Height: 10
 (x1,y1)=(20,20), (x2,y2)=(35,20), (x3,y3)=(35,30), (x4,y4)=(20,30)
[Square][Rectangle] x: 30, y: 30, Width: 15, Height: 15
 (x1,y1)=(30,30), (x2,y2)=(45,30), (x3,y3)=(45,45), (x4,y4)=(30,45)
[Square][Rectangle] x: 30, y: 30, Width: 15, Height: 15
 (x1,y1)=(30,30), (x2,y2)=(45,30), (x3,y3)=(45,45), (x4,y4)=(30,45)
```

```
Draw Quadrangle:x: 10, y: 10, Width: 10, Height: 15
 (x1,y1)=(15,10), (x2,y2)=(20,20), (x3,y3)=(15,25), (x4,y4)=(10,15)
Draw Rectangle:[Rectangle] x: 20, y: 20, Width: 15, Height: 10
 (x1,y1)=(20,20), (x2,y2)=(35,20), (x3,y3)=(35,30), (x4,y4)=(20,30)
Draw Square:[Square][Rectangle] x: 30, y: 30, Width: 15, Height: 15
 (x1,y1)=(30,30), (x2,y2)=(45,30), (x3,y3)=(45,45), (x4,y4)=(30,45)
Draw Square:[Square][Rectangle] x: 30, y: 30, Width: 15, Height: 15
 (x1,y1)=(30,30), (x2,y2)=(45,30), (x3,y3)=(45,45), (x4,y4)=(30,45)
```