



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

PROGRAMMATION ET TP

T1A

RÉSEAU ET SÉCURITÉ

S17 Héritage et polymorphisme

ROTEN MARC

2017/2018

Table des matières

0	Introduction	2
1	Exercice 1 Les entêtes	2
2	Exercice 2 Les bidons	3
2.1	Ex2 a classe Bidon	3
2.2	Ex2 b classe EcoBidon	4
2.3	Ex2 c Question théorique	5
2.3.1	Plusieurs couches de réservoirs?	5
2.3.2	cycles de réservoirs	5
2.4	Ex2 d Comparez les deux méthodes	5
3	Ex 3 le Cercle	5
3.1	Version A	6
3.2	Version B	7
4	Conclusion	8

0 Introduction

Avec ce travail, on va s'occuper de l'héritage et du polymorphisme qui est indispensable dans le langage de programmation java.

1 Exercice 1 Les entêtes

- 1 Identifier les relations ("est un", "possède un") qui existent entre les concepts suivants :

Chien Oeil Collier Chat Animal Oiseau Tête Organe Aile Patte Pigeon

Déclarer l'entête des 11 classes Java correspondantes avec juste les attributs concernés.

[FACULTATIF] Esquisser un diagramme de classes (nom des classes, attributs et héritage).


```
public class Animal {
    Tete t;
}
public class Tete extends Organe {
    Oeil leftEye;
    Oeil RightEye;
}
public class Oiseau extends Animal {
    Patte rightHand;
    Patte leftHand;
    Aile leftWing;
    Aile rightWing;
}
public class Chat extends Animal {
    Collier c;
    Patte p1;
    Patte p2;
    Patte p3;
    Patte p4;
}
public class Collier {
}
public class Oeil extends Organe {
}
public class Pigeon extends Oiseau {
}
public class Aile extends Organe {
}
public class Organe {
}
public class Patte extends Organe {
}
```

2 Exercice 2 Les bidons

2^a On veut définir une sorte d'objet "*bidon d'eau*", offrant les fonctionnalités suivantes (spécification ci-contre) :

- *créer* un bidon en indiquant sa capacité sous forme d'un nombre entier de litres; le bidon créé sera vide;
- *remplir* un bidon à la rivière, ou le vider entièrement par terre;
- *transvaser* un bidon dans un autre : on verse autant de liquide que possible, donc jusqu'à ce que l'un soit plein (il y aura un reste dans l'autre), ou que l'autre soit vide;
- *consulter* le volume de liquide actuellement présent.

```
public class Bidon {  
    //PRE: capacity > 0  
    public Bidon(int capacity);  
    //POST: it must be full  
    public void fill();  
    //POST: it must be empty  
    public void empty();  
    public void transferFrom(Bidon b);  
    public int contents();  
    @Override  
    public String toString();  
}
```



En respectant strictement le principe d'encapsulation, compléter la classe Bidon.



2^b Modélisons un *bidon écologique* comme une sorte spécialisée de bidon. Dès la création,

un EcoBidon est associé à un bidon "réservoir" qui servira de tampon : autant que possible, l'éco-bidon se remplit (resp. vide) depuis (*dans*) son réservoir. Les éco-bidons sont responsables de ne pas laisser à sec le réservoir associé (ils lui ordonnent de se remplir si nécessaire). Implémenter cette sous-classe en redéfinissant fill() et empty(). Un petit programme test (fourni) précise le comportement attendu.

```
public class EcoBidon extends Bidon {  
    ...  
    public EcoBidon(int capacity,  
                    Bidon tank) {...  
}
```



2^c Expliquer si votre solution (2^b) offre les deux fonctionnalités suivantes :

- permettre de gérer plusieurs "couches" de réservoirs ($a \rightarrow b \rightarrow c$)
- empêcher des "cycles" de réservoirs ($a \rightarrow b \rightarrow a$)

2^d Comparer l'effet de ces deux méthodes collect().

- Le bidon retourné sera-t-il plein ?
- Les bidons du tableau seront-ils vides après l'appel ?

Réfléchissez bien !...

```
static Bidon collect1(Bidon[] t) {  
    int total=0;  
    for(Bidon b:t) total+=b.contents();  
    Bidon res=new Bidon(total);  
    for(Bidon b:t) res.transferFrom(b);  
    return res;  
}
```

```
static Bidon collect2(Bidon[] t) {  
    int total=0;  
    for(Bidon b:t) {  
        total+=b.contents();  
        b.empty();  
    }  
    Bidon res=new Bidon(total);  
    res.fill();  
    return res;  
}
```

2.1 Ex2 a classe Bidon

```
public class Bidon {  
    private final int capacity;  
    private int contents;  
  
    //constructeur  
    public Bidon(int cap) {  
        capacity=cap;  
    }  
    //remplissage  
    public void fill() {  
        contents=capacity;  
    }  
    public void empty() {
```

```

        contents=0;
    }
    public void transferFrom(Bidon b) {
        int freespace = capacity - contents;
        if(freespace >= b.contents) {
            contents+= b.contents;
            b.empty();
        } else if(freespace >0 && freespace < b.contents) {
            contents+= freespace;
            b.setContents(b.contents-freespace);
        }
    }
    public void setContents(int c){
        this.contents = c;
    }
    public int contents() {
        return contents;
    }
    public int capacity() {
        return capacity;
    }
    @Override
    public String toString() {
        return "Capacity: "+capacity+", Contents: "+contents();
    }
}

```

2.2 Ex2 b classe EcoBidon

```

    public class EcoBidon extends Bidon {
    public static void main(String[] args) {
        Bidon myBidon = new Bidon(40);
        myBidon.fill();
        //capacite fixee arbitrairement a 5
        EcoBidon MyEcoBidon = new EcoBidon (5,myBidon);
        //si l'ecobidon est vide on le remplit avec le bidon
        if(MyEcoBidon.capacity()==0) {
            MyEcoBidon.transferFrom(myBidon);
        }
        // si le bidon est vide on le remplit
        if(myBidon.capacity()==0) {
            myBidon.fill();
        }
    }
    public EcoBidon(int capacity,Bidon tank) {
        super(capacity);
    }
}

```

}

2.3 Ex2 c Question théorique

2.3.1 Plusieurs couches de réservoirs ?

non, on ne peut pas gérer plusieurs couches de réservoir telles que A->B->C, il faudrait créer un tableau pour mettre en lien les différents écoBidons, ce qui n'est pas le cas avec la solution de l'exercice 2B

2.3.2 cycles de réservoirs

non, avec la solution proposée en 2b, on ne peut réaliser de cycle de réservoir de type A->B->A, il y aurait un conflit avec les contenus des ecobidons

2.4 Ex2 d Comparez les deux méthodes

Pour la méthode Collect1, on ne connaît pas si les bidons passés en paramètres dans le tableau sont pleins ou non, donc dans la méthode transferFrom, le bidon retourné sera plein si tous les bidons passés en paramètres sont pleins. Les bidons passés en paramètres seront vidés.

Pour la méthode collect2, à l'avant-dernière ligne, on réalise res.fill(), le bidon retourné sera donc plein. Les bidons passés en paramètres seront vidés.

3 Ex 3 le Cercle

3 On dispose de la classe Point ci-contre. On souhaite réaliser une classe Circle avec ces fonctionnalités :

- un constructeur recevant en argument les coordonnées du centre du cercle et son rayon
- moveCenter() pour déplacer les coordonnées du centre du cercle (incréments dx et dy)
- changeRadius() pour modifier le rayon du cercle
- center() qui fournit en résultat un objet de type Point correspondant au centre du cercle
- redéfinir la méthode toString() afin qu'elle retourne une chaîne de caractères contenant les coordonnées du centre du cercle ainsi que son rayon

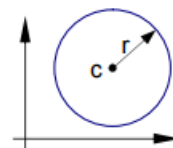
```
public class Point {  
    private double x, y;  
    public Point (double x, double y) {  
        this.x = x; this.y = y;  
    }  
    public void move(double dx, double dy) {  
        x += dx; y += dy;  
    }  
    @Override  
    public String toString() {  
        return "Point at (" + x + ", " + y + ")";  
    }  
    public double getX() { return x; }  
    public double getY() { return y; }  
}
```

Implémenter deux variantes de cette classe Circle :

- a) comme classe dérivée (sous-classe) de Point ⇒ *par héritage*
- b) avec un attribut de type Point ⇒ *par composition*

Quelle version vous semble préférable ?

Attention : Porter une attention particulière aux deux manières différentes de définir le comportement de la méthode center(), c'est plus important qu'il n'y paraît...



3.1 Version A

classe CircleVA

```
public class CircleVa extends Point {
private double myRayon;
public CircleVa(double posX, double posY, double rayon) {
    super(posX, posY);
    this.myRayon=rayon;
}
public void moveCenter(double dx, double dy) {
    super.move(dx, dy);
}
public void changeRadius(double dr) {
    this.myRayon=dr;
}
public Point Center() {
    return new Point(getX(), getY());
}
public String toString() {
    return "le cercle possede un rayon de "+myRayon+" cm. Son
           centre se trouve en "
           +this.Center().getX()+"/"+this.Center().getY();
}
}
```

classe TestCircle

```
public class TestCircleVa {
public static void main(String[] args) {
    System.out.println("creation du cercle");
    CircleVa circle1= new CircleVa(2,3, 5);
    System.out.println(circle1.toString());
    System.out.println("deplacement de 2 vers la droite et de 1
           vers le bas");
    circle1.move(2, -1);
    System.out.println(circle1.toString());
    System.out.println("rayon de 10");
    circle1.changeRadius(10);
    System.out.println(circle1.toString());
}
}
```

```
Problems @ Javadoc Declaration Console Debug
<terminated> TestCircleVa [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (18 déc. 2017 à 16:02:34)
création du cercle
le cercle possède un rayon de 5.0 cm. Son centre se trouve en 2.0/3.0
déplacement de 2 vers la droite et de 1 vers le bas
le cercle possède un rayon de 5.0 cm. Son centre se trouve en 4.0/2.0
rayon de 10
le cercle possède un rayon de 10.0 cm. Son centre se trouve en 4.0/2.0
```

3.2 Version B

classe circle

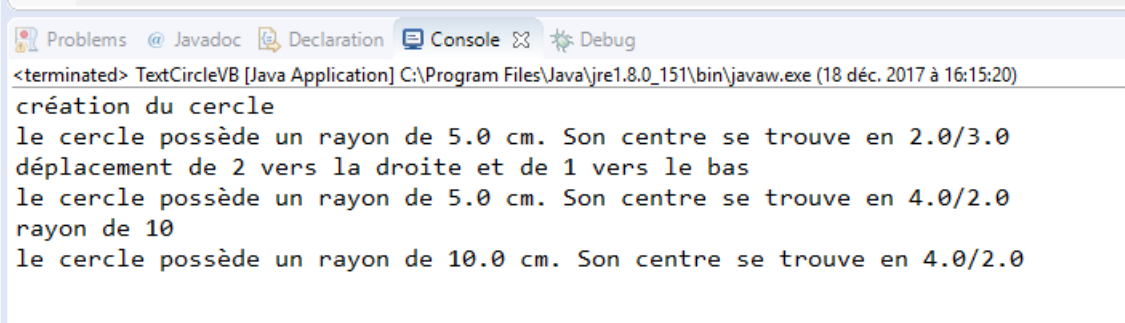
```
import s17.Circle.Point;
public class CircleVB extends Point {
    private double myRayon;
    private Point Center;
    public CircleVB(double posX, double posY, double rayon) {
        super(posX, posY);
        this.Center = new Point(posX, posY);
        this.myRayon=rayon;
    }
    public void moveCenter(double dx, double dy) {
        Center.move(dx, dy);
    }
    public void changeRadius(double dr) {
        this.myRayon=dr;
    }
    public Point Center() {
        return new Point(getX(), getY());
    }
    public String toString() {
        return "le cercle possède un rayon de "+myRayon+" cm. Son
            centre se trouve en "
            +this.Center().getX()+"/"+this.Center().getY();
    }
}
```

classe test

```
public class TextCircleVB {
    public static void main(String[] args) {
        System.out.println("creation du cercle");
        CircleVB circle1= new CircleVB(2,3, 5);
        System.out.println(circle1.toString());
        System.out.println("deplacement de 2 vers la droite et de 1
            vers le bas");
    }
}
```

```
        circle1.move(2, -1);
        System.out.println(circle1.toString());
        System.out.println("rayon de 10");
        circle1.changeRadius(10);
        System.out.println(circle1.toString());
    }
}
```

Résultat à la console.



```
<terminated> TextCircleVB [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (18 déc. 2017 à 16:15:20)
création du cercle
le cercle possède un rayon de 5.0 cm. Son centre se trouve en 2.0/3.0
déplacement de 2 vers la droite et de 1 vers le bas
le cercle possède un rayon de 5.0 cm. Son centre se trouve en 4.0/2.0
rayon de 10
le cercle possède un rayon de 10.0 cm. Son centre se trouve en 4.0/2.0
```

les deux versions sont équivalentes.

4 Conclusion

Ce travail sur l'héritage et le polymorphisme m'a permis de mieux comprendre certains aspects de la programmation JAVA.