TD2 : Héritage
Correction

Exercice 1. Exécution de code

Que fournit l'exécution du code suivant :

```
class A {
    public A (int n) {
        System.out.println ("Entree Constr A - n=" + this.n + " p=" + this.p);
        this.n = n;
        System.out.println ("Sortie Constr A - n=" + this.n + " p=" + this.p);
    }
    public int n;
    public int p=10;
}

class B extends A {
    public B (int n, int p) {
        super (n);
        System.out.println ("Entree Constr B - n=" + this.n + " p=" + this.p + " q=" + this.q);
        this.p = p;
        this.q = 2*n;
        System.out.println ("Sortie Constr B - n=" + this.n + " p=" + this.p + " q=" + this.q);
    }
    public int q=25;
}

public class Ex1 {
    public static void main (String args[]){
        A a = new A(5);
        B b = new B(5, 3);
    }
}
```

Correction

La création de l'objet a est habituelle : elle suit les trois étapes (initialisation par défaut, initialisation explicite et exécution du constructeur) décrites dans le cours et dans l'exercice 2 du TD1. Elle va donc générer les affichages suivants :

```
Entree Constr A - n=0 p=10
Sortie Constr A - n=5 p=10
```

Comme la classe B dérive de A, l'instanciation de l'objet b suit des étapes supplémentaires, que nous rappelons ici :

- Allocation mémoire pour un objet de type B (donc de tous les attributs de B, ce qui comprend ceux hérités de A)
- Initialisation par défaut de tous les attributs
- Initialisation explicite des attributs hérités de A
- Exécution du corps du constructeur de A
- Initialisation explicite des attributs de B
- Exécution du corps du constructeur de B

Détaillons les valeurs prises par les attributs de b lors de B b = **new** B(5, 3);

	Initialisation par défaut	Initialisation explicite des attributs hérités de A	Exécution du corps du constructeur de A	Initialisation explicite des attributs de B	Exécution du corps du constructeur de B
n	0	0	5	5	5
p	0	10	10	10	3
q	0	0	0	25	10

Les lignes affichées seront donc les suivantes :

```
Entree Constr A - n=0 p=10
Sortie Constr A - n=5 p=10
Entree Constr B - n=5 p=10 q=25
Sortie Constr B - n=5 p=3 q=10
```

Remarque L'exécution du constructeur de A ne se produit que grâce à l'appel **super**(n) présent dans le constructeur de B.

Exercice 2. Héritage et redéfinition

Que fournit l'exécution du code suivant :

```
class A {
    public void affiche() {
        System.out.println ("Je suis un A") ;
    }
}
class B extends A {}
class C extends A {
    public void affiche() {
        System.out.println ("Je suis un C");
    }
}
class D extends C {
    public void affiche() {
        System.out.println ("Je suis un D");
    }
}
class E extends B {}
class F extends C {}

public class Ex2 {
    public static void main (String arg[]) {
        A a = new A() ; a.affiche();
        B b = new B() ; b.affiche();
        C c = new C() ; c.affiche();
        D d = new D() ; d.affiche();
        E e = new E() ; e.affiche();
        F f = new F() ; f.affiche();
    }
}
```

Correction
Si une méthode n'est pas redéfinie dans une classe fille, c'est la méthode de la classe mère qui est appelée. On obtient donc :

```
Je suis un A
Je suis un A
Je suis un C
Je suis un D
Je suis un A
Je suis un C
```

Exercice 3. Polymorphisme

```
class A { public void affiche() {System.out.println ("Je suis un A");}}
class B extends A {}
class C extends A { public void affiche() { System.out.println ("Je suis un C");}}
class D extends C { public void affiche() { System.out.println ("Je suis un D");}}
class E extends B {}
class F extends C {}

public class Ex3 {
    public static void main (String arg[]) {
        A a = new A(); a.affiche();
```

```
        B b = new B(); b.affiche();
        a = b;
        a.affiche();
        C c = new C(); c.affiche();
        a = c; a.affiche();
        D d = new D(); d.affiche();
        a = d; a.affiche();
        c = d; c.affiche();
        E e = new E(); e.affiche();
        a = e; a.affiche();
        b = e; b.affiche();
        F f = new F(); f.affiche();
        a = f; a.affiche();
        c = f; c.affiche();
    }
}
```

a. Que fournit l'exécution du code ci-dessus ?

Correction
La méthode appelée dépend du type de l'objet, et non du type de la référence. Autrement dit, à la ligne a = c; a. affiche (), c'est la méthode de la classe C qui est appelée car a devient une référence vers l'objet c qui est de type C. Le type de la référence, ici définie comme un A, n'intervient pas dans le choix de la méthode appelée.

```
Je suis un A
Je suis un A
Je suis un A
Je suis un C
Je suis un C
Je suis un D
Je suis un D
Je suis un D
Je suis un A
Je suis un A
Je suis un A
Je suis un C
Je suis un C
Je suis un C
```

b. Certaines possibilités d'affectation entre objets des types classes A, B, C, D, E et F ne figurent pas dans le programme ci-dessus. Pourquoi ? Donnez un exemple d'affectation entraînant une erreur de compilation.

Correction
On peut écrire a = c; car C dérive de A. Un objet de type C *est* donc aussi un objet de type A. L'inverse n'est pas vrai : un objet de type A n'est pas un objet de type C.

Exercice 4. Limites du polymorphisme

Soit les classes Point et PointNom ainsi définies :

```
class Point {
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
    public static boolean identiques(Point a, Point b) {
        return ( (a.x==b.x) && (a.y==b.y) );
    }
    public boolean identique(Point a) {
        return ( (a.x==x) && (a.y==y) );
    }
    private int x, y;
}
```

```
class PointNom extends Point {
    PointNom (int x, int y, char nom) {
        super (x, y);
        this.nom = nom;
    }

    private char nom;
}

public class Ex4 {
    public static void main (String args[]) {
        Point p = new Point (2, 4);
        PointNom pn1 = new PointNom (2, 4, 'A');
        PointNom pn2 = new PointNom (2, 4, 'B');
        System.out.println (pn1.identique(pn2));
        System.out.println (p.identique(pn1));
        System.out.println (pn1.identique(p));
        System.out.println (Point.identiques(pn1, pn2));
        System.out.println (pn1.identiques(pn1, pn2));

        Point pn1p = pn1; Point pn2p = pn2;
        System.out.println (PointNom.identiques(pn1p, pn2p)); }}
```

a. Quels résultats fournit ce programme ? Expliciter les conversions mises en jeu et les règles utilisées pour traiter les différents appels de méthodes.

Correction
PointNom ne redéfinit pas les méthodes identique et identiques, c'est donc celles de la classe Point qui sont appelées. Comme elles ne comparent que les coordonnées des points et non leurs noms, l'exécution donne :

```
true
true
true
true
true
true
```

b. Doter la classe PointNom d'une méthode statique identiques et d'une méthode identique fournissant toutes les deux la valeur true lorsque les deux points concernés ont à la fois mêmes coordonnées et même nom. Quels résultats fournira alors le programme précédent ? Quelles seront les conversions mises en jeu et les règles utilisées ?

Correction

```
    public static boolean identiques (PointNom a, PointNom b) {
        return (Point.identiques(a,b) && (a.nom == b.nom));
    }
    public boolean identique (PointNom a) {
        return (super.identique(a) && (a.nom == nom));
    }
```

L'exécution donne :

```
false
true
true
true
false
true
```

Le point délicat est soulevé par le dernier appel PointNom.identiques(pn1p, pn2p). On a vu dans l'exercice précédent que la méthode appelée dépend de la classe de l'objet, et non de sa référence. Cela s'applique

pour déterminer la méthode de quelle classe est appeler. Mais pour choisir la méthode qui convient (parmi les multiples surcharges possibles), Java détermine laquelle est la mieux adaptée en fonction du type des arguments. Ici, Java a le choix entre
— la méthode identiques(Point a, Point b) de classe Point
— la méthode identiques(PointNom a, PointNom b) de la classe PointNom
Ici, nous avons explicitement casté les PointNom en Point, c'est donc la première qui sera appelée.

Exercice 5. Classe Cercle

On suppose donnée la classe Point suivante :

```
class Point {
    public Point (double x, double y) {this.x = x; this.y = y;}
    public void deplace (double dx, double dy) {x += dx; y += dy;}
    public void affiche() {System.out.println("Point de coordonnees " + x + " " + y);}
    public double getX() {return x;}
    public double getY() {return y;}
    private double x,y;
}
```

On souhaite réaliser une classe Cercle disposant des méthodes suivantes :
— un constructeur avec les coordonnées du centre et un rayon
— deplace qui déplace le centre du cercle
— changeRayon
— getCentre
— affiche

a. Définir la classe Cercle comme ayant un objet membre Point

Correction

```
class Cercle {

    public Cercle(double x, double y, double r) {
        this.centre = new Point(x, y);
        this.r = r;
    }

    public Point getCentre() {
        return centre;
    }

    public void affiche() {
        System.out.println("Cercle de centre (" + centre.getX() + "," + centre.getY()
        + ") et de rayon " + r);
    }

    public void deplace(double dx, double dy) {
        centre.deplace(dx, dy);
    }

    public void changeRayon(double r){
        this.r = r;
    }

    private Point centre;
    private double r;
}
```

b. Définir la classe Cercle comme dérivée de Point

Correction

```java
class Cercle2 extends Point {

    public Cercle2(double x, double y, double r) {
        super(x,y);
        this.r = r;
    }

    public Point getCentre() {
        return new Point(this.getX(), this.getY());
    }

    public void affiche() {
        System.out.println("Cercle de centre (" + getX() + "," + getY() + ") et de rayon " + r);
    }

    public void changeRayon(double r){
        this.r = r;
    }

    private double r;
}
```

Exercice 6. Structures algébriques

Hiérarchie des structures algébriques de l'ensemble $\mathbb{Z}$

On souhaite représenter les différentes structures algébriques (ensemble, groupe, anneau) de l'ensemble $\mathbb{Z}$, grâce à la programmation par objets.

a. Proposer une hiérarchie de classes pour représenter les différentes structures de l'ensemble $\mathbb{Z}$, dont les éléments seront représentés par des `int` : ensemble, groupe, anneau.

Correction
On rappelle la définition de ces structures.
$(G, +)$ est un groupe ssi
— loi interne : $\forall x, y \in G,\ x + y \in G$
— associativité : $\forall x, y, z \in G,\ (x + y) + z = x + (y + z)$
— élément neutre : $\exists e \in G,\ \forall x \in G,\ e + x = x + e = x$
$(A, +, \cdot)$ est un anneau unitaire ssi
— loi interne : $(A, +)$ est un groupe commutatif
— associativité de $\cdot$: $\forall x, y, z \in A,\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$
— distributivité : $\forall x, y, z \in A,\ x \cdot (y + z) = x \cdot y + x \cdot z$
— élément neutre de $\cdot$: $\exists n \in A,\ \forall x \in A,\ n \cdot x = x \cdot n = x$

```java
interface Set {
        public static boolean is_element(Object a);
}

interface Group {
        public abstract int add(int a, int b);
        public abstract int neg(int a);
        public abstract int sub(int a, int b);
        public abstract boolean is_g_neutral_element(int a);
}

interface Ring extends Group {
        public abstract int mult(int a, int b);
        public abstract int div(int a, int b);
        public abstract int inv(int a);
        public abstract boolean is_invertible(int a);
        public abstract boolean is_r_neutral_element(int a);
}
```

b. Écrire en Java les implémentations de ces classes.

Correction

```
class ZSet implements Set {
    public static boolean is_element(Object a) {
        return a instanceof Integer;
    }
}

class ZGroup extends Zset implements Group {
        public int add(int a, int b) {return a + b;}
        public int neg(int a) {return -a;}
        public int sub(int a, int b) {return add(a,neg(b));}
        public boolean is_g_neutral_element(int a) {return (a==0);}
}

class ZRing extends ZGroup implements Ring {
        public int mult(int a, int b) {return a*b;}
        public int div(int a, int b) { return 0; }
        public int inv(int a) { return 0; }
        public boolean is_invertible(int a) {return (a == 1);}
        public boolean is_r_neutral_element(int a) {return (a == 1);}
}
```

Généralisation

On souhaite maintenant aussi représenter les différentes structures algébriques de l'ensemble $\mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$: ensemble, groupe additif, anneau, corps.

c. Proposer une hiérarchie de classes basée sur la hiérarchie précédente, en détaillant les nouvelles classes et leurs nouveaux membres.

d. Y-a-t-il plusieurs possibilités ? Si oui, laquelle préférez-vous ?

e. Comment isoler les structures communes de groupe, d'anneau, de corps, propres à ces ensembles ?

f. Proposer une nouvelle implémentation basée sur les classes abstraites ou sur les interfaces.

Espace vectoriel et anneau des polynômes $\mathbb{Z}/p\mathbb{Z}[X]$

On souhaite maintenant représenter les polynômes de degré inférieur à un entier d, à coefficients dans un corps $\mathbb{Z}/p\mathbb{Z}$, avec p premier : $(\mathbb{Z}/p\mathbb{Z}[X])_d$. On rappelle que cet ensemble a une structure d'espace vectoriel sur $\mathbb{Z}/p\mathbb{Z}$.

g. Quelle structuration de classe allez-vous employer (interface, héritage, classe interne, objet membre...) pour cette nouvelle classe.

h. Écrire cette classe polynôme.

i. Peut-on utiliser votre classe pour représenter indifféremment les polynômes à coefficients dans $\mathbb{Z}$ et dans $\mathbb{Z}/p\mathbb{Z}$. Faire les modifications nécessaires le cas échéant.

Pour aller plus loin : structures génériques et polymorphisme

On rappelle que l'ensemble des polynômes de degré inférieur à d, à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ possède également la structure d'anneau, de part sa loi de multiplication $\times$. On souhaiterait donc maintenant illustrer le fait que les deux ensembles $Z/p\mathbb{Z}$ et $(Z/p\mathbb{Z}[X])_d$ sont des anneaux.

j. Peut-on utiliser le même procédé que pour la question f ?

k. Proposer une solution. (Indication : on pourra essayer de créer une classe abstraite `Element` de laquelle dériveront des classes représentant les ensembles $\mathbb{Z}/n\mathbb{Z}$ et les polynômes $(\mathbb{Z}/p\mathbb{Z}[X])_d$ pour utiliser le polymorphisme.

Remarque : on verra bientôt en cours que la programmation générique permet de résoudre plus élégamment cette difficulté.