MOOC Intro POO Java
Corriges semaine 4

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 11 : Affichage et comparaison d'objets

Les méthodes `toString` et `equals` redéfinissent (masquent) celles héritées de `Object`. Les tests fournis montrent qu'elles fonctionnent de façon polymorphique (affichage de `rect2` et ligne commentée .2 dans la séquence "Test 3 :"). Les points auxquels il fallait être attentifs sont commentés dans la solution ci-dessous :

```
class ToStringEq
{

    public static void main(String[] args)
        {
            System.out.println("Test 1 :");
            Rectangle rect = new Rectangle(12.5, 4.0);
            System.out.println(rect);
            System.out.println();

            System.out.println("Test 2: ");
            // le type de rect1 est RectangleColore
            // l'objet contenu dans rect1 est de type RectangleColore
            RectangleColore rect1 = new RectangleColore(12.5, 4.0, "rouge");
            System.out.println(rect1);
            System.out.println();

            System.out.println("Test 3 :");

            // le type de rect2 est Rectangle
            // l'objet contenu dans rect2 est de type RectangleColore
            Rectangle  rect2 = new RectangleColore(25.0/2, 8.0/2, new String("rouge"));
            System.out.println(rect2);

            System.out.println (rect1.equals(rect2)); // 1.
            System.out.println (rect2.equals(rect1)); // 2.
            System.out.println(rect1.equals(null));   // 3.
            System.out.println (rect.equals(rect1));  // 4.
            System.out.println (rect1.equals(rect));  // 5.
        }
}


class Rectangle
{
    private double largeur;
    private double hauteur;

    public Rectangle(double uneLargeur, double uneHauteur)
        {
            largeur = uneLargeur;
            hauteur = uneHauteur;

        }

    public boolean equals(Object autreRect)
        {
            // permet de passer correctement la ligne commentée .3 du test 3:
            if (autreRect == null)
                return false;
            else
                // garantit que l'on compare bien des
                // objet de même classe
                if (autreRect.getClass() != getClass()){
                    return false;
                }

                else
```

```
                {
                    // procède à la comparaison attribut par
                    // attribut
                    return (
                        // Ne pas oublier le transtypage ici
                        (largeur == ((Rectangle)autreRect).largeur)
                        &&
                        hauteur == ((Rectangle)autreRect).hauteur);
                }

        }
    public String toString()
        {
            return "Rectangle : \n "
                + "largeur = " + largeur + "\n hauteur = " + hauteur;
        }

}


class RectangleColore  extends Rectangle
{
    private String couleur;

    public RectangleColore(double uneLargeur, double uneHauteur, String uneCouleur)
        {
            super(uneLargeur, uneHauteur);
            couleur = uneCouleur;
        }


    public boolean equals(Object autreRectColore)
        {
            if (autreRectColore == null){
                return false;
            }

            else if (autreRectColore.getClass() != getClass())
            {
                return false;
            }

            else
            {
                // Réutilisation du equals de la super-classe
                // pour éviter toute duplication de code
                return (super.equals(autreRectColore) &&
                        couleur.equals(((RectangleColore)autreRectColore).couleur)
                        );
                    }


        }

    public String toString()
        {
            // Réutilisation de toString de la super-classe
            // pour éviter toute duplication de code
            return (super.toString() + "\n couleur = " + couleur);

        }

}
```

Exercice 12 : Tour de cartes

Le code complet vous est donné ci-dessous:

```
/**
 *  Une petite classe utilitaire pour commencer
 */
class Couleur {
    private char valeur;

    public Couleur(char c) {
        valeur = c;
    }

    public void afficher() {
        this.afficher(false);
    }

    public void afficher(boolean feminin) {
        switch (valeur) {
        case 'r':
            System.out.println("rouge");
            break;
        case 'v':
            System.out.print("vert");
            if (feminin) {
                System.out.println("e");
            }
            break;
        case 'b':
            System.out.print("bleu");
            if (feminin) {
                System.out.println("e");
            }
            break;
        case 'B':
            System.out.print("blanc");
            if (feminin) {
                System.out.println("he");
            }
            break;
        case 'n':
            System.out.print("noir");
            if (feminin) {
                System.out.println("e");
            }
            break;
        }
    }
}

// ------------------------------------------------------------------------
// puis.. les classes principales

abstract class Carte {

    private int cout;

    public Carte() {
        cout = 0;
    }

    public Carte(int cout) {
        this.cout = cout;
    }

    public abstract void afficher();
}

// ------------------------------------------------------------------------

class Terrain extends Carte {
    private Couleur couleur;

    public Terrain(char c) {
```

```
        couleur = new Couleur(c);
        System.out.println("Un nouveau terrain.");
    }

    public void afficher() {
        System.out.print("Un terrain ");
        couleur.afficher();
        System.out.println();
    }

}

// ---------------------------------------------------------------------

class Creature extends Carte {
    private String nom;
    private int attaque;
    private int defense;

    public Creature(int cout, String nom, int attaque, int defense) {
        super(cout);
        this.nom = nom;
        this.attaque = attaque;
        this.defense = defense;
        System.out.println("Une nouvelle créature.");
    }

    public void afficher() {
        System.out.println("Une créature " + nom + " " + attaque + "/"
                + defense + " ");
    }
}


// ---------------------------------------------------------------------

class Sortilege extends Carte {
    private String nom;
    private String description;

    public Sortilege(int cout, String nom, String desc) {
        super(cout);
        this.nom = nom;
        this.description = desc;
        System.out.println("Un sortilège de plus.");
    }

    public void afficher() {
        System.out.println("Un sortilège " + nom + " ");
    }

}

// ---------------------------------------------------------------------

class Jeu {
    private int nombreCartes;
    private Carte[] cartes;

    public Jeu(int nb) {
        nombreCartes = nb;
        cartes = new Carte[nb];
        System.out.println("On change de main");
    }

    /**
     * Joue une carte après l'autre
     */
    public void joue() {
        System.out.println("Je joue une carte...");
        int i = 0;
        while ((cartes[i] == null) && i < nombreCartes) {
            i++;
        }
        if ((i < nombreCartes) && (cartes[i] != null)) {
            System.out.println("La carte jouée est :");
```

```
            cartes[i].afficher();
            cartes[i] = null;
        } else {
            System.out.println("Plus de cartes");
        }
    }

    /**
     * Ajoute une carte à la collection
     */
    public void piocher(Carte carte) {
        int i = 0;
        while ((i < nombreCartes) && (cartes[i] != null)) {
            i++;
        }
        if (i < nombreCartes) {
            cartes[i] = carte;
        } else {
            System.out.println("Nombre maximal de cartes atteint");
        }
    }

    public void afficher() {
        for (int i = 0; i < nombreCartes; ++i) {
            if (cartes[i] != null) {
                cartes[i].afficher();
            }
        }
    }

}

// ---------------------------------------------------------------------

class Magic {
    public static void main(String[] args) {
        Jeu maMain = new Jeu(10);

        maMain.piocher(new Terrain('b'));
        maMain.piocher(new Creature(6, "Golem", 4, 6));
        maMain.piocher(new Sortilege(1, "Croissance Gigantesque",
                "La créature ciblée gagne +3/+3 jusqu'à la fin du tour"));

        System.out.println("Là, j'ai en stock :");
        maMain.afficher();
        maMain.joue();
    }

}
```

Exercice 13 : Analyse de programme

Les constructeurs (constructeurs de copie compris) ne sont pas polymorphiques. Le programme mettra donc dans la collection des copies d'objets de type Forme et non pas des copies des objets contenus dans tabForm. Le programme affichera donc:

```
Une forme rouge
Une forme jaune
```

Pour corriger le programme, il faut utiliser des méthodes de copie polymorphiques. Ce sont les méthodes clone dans le code corrigé suivant :

```
class Polymorph{

    public static void main(String[] args){

        Forme[] tabFormes = {new Cercle("rouge"),
                new Triangle("jaune")
        };

        Collect formes = new Collect(10);

        // Une collection de formes
        // contenant une copie des objets definis
        // dans le tableau tabFormes
        for (int i = 0; i < tabFormes.length; ++i)
            formes.add(tabFormes[i].clone());
        formes.dessine();
    }
}



class Forme {

    private String couleur;

    public Forme(String uneCouleur) {
        couleur = uneCouleur;
    }

    public Forme clone() {
        return new Forme(this);
    }

    public Forme(Forme other) {
        this.couleur = other.couleur;
    }
    public void dessine(){
        System.out.println("Une forme " + couleur);
    }
}



class Triangle extends Forme {

    public Triangle(String uneCouleur){
        super(uneCouleur);
    }

    public Triangle(Triangle autreTriangle){
        super(autreTriangle);
    }

    public Triangle clone(){
        return new Triangle(this);
    }

    public void dessine(){
        super.dessine();
        System.out.println("toute pointue");
    }
}
```

```
class Cercle extends Forme{

    public Cercle(String uneCouleur){
        super(uneCouleur);
    }

    public Cercle(Cercle autreCercle){
        super(autreCercle);
    }

    public Cercle clone(){
        return new Cercle(this);
    }



    public void dessine(){
        super.dessine();
        System.out.println("toute ronde");
    }
}


class Collect{

    private Forme collect[];

    private int index;

    public Collect(int indexMax){
        collect = new Forme[indexMax];
        index = -1;
    }

    public void add(Forme a){
        if (index < collect.length - 1){
            ++ index;
            collect[index] = a;
        }

    }
    public void dessine(){
        for (int i = 0; i <= index; ++i){
            collect[i].dessine();
        }
    }
}
```

Exercice 14 : Cryptographie

```
import java.util.Random;


class Utils {
    // genere un entier entre 1 et max (compris)
    public static int randomInt(int max) {
        Random r = new Random();
        int val = r.nextInt();
        val = Math.abs(val);
        val = val % max;
        val += 1;
        return val;
    }
}

class Secret {

    public static void main(String[]  args){
        String message = "COURAGEFUYONS";
        String cryptage;

        // TEST A CLE
        Code acle1 = new ACle("a cle", "EQUINOXE");
        System.out.print("Avec le code : " );
        acle1.affiche();
        cryptage  = acle1.code(message);
        System.out.print("Codage de " + message + " : ");
        System.out.println(cryptage);
        System.out.print("Decodage de " + cryptage + " : ");
        System.out.println(acle1.decode(cryptage));
        System.out.println("-----------------------------------");
        System.out.println();
        // FIN TEST A CLE

        // TEST A CLE ALEATOIRE
        Code acle2 = new ACleAleatoire(5);
        System.out.print("Avec le code : " );
        acle2.affiche();
        cryptage  = acle2.code(message);
        System.out.print("Codage de " + message + " : ");
        System.out.println(cryptage);
        System.out.print("Decodage de " + cryptage + " : ");
        System.out.println(acle2.decode(cryptage));
        System.out.println("-----------------------------------");
        System.out.println();
        // FIN TEST A CLE ALEATOIRE

        // TEST CESAR

        Code cesar1 = new Cesar("Cesar", 5);
        System.out.print("Avec le code : " );
        cesar1.affiche();
        cryptage = cesar1.code(message);
        System.out.print("Codage de " + message + " : ");
        System.out.println(cryptage);
        System.out.print("Decodage de " + cryptage + " : ");
        System.out.println(cesar1.decode(cryptage));
        System.out.println("-----------------------------------");
        System.out.println();
        // FIN TEST CESAR

        // TEST CODAGES
        System.out.println("Test CODAGES: ");
        System.out.println("------------- ");
        System.out.println();


        Code[] tab = {   // Decommentez la ligne suivante
                // si vous avez fait la classe Cesar
                new Cesar("cesar", 5),
                new ACle("a cle", "EQUINOXE") ,
                new ACleAleatoire(5),
                new ACleAleatoire(10)};
```

```
        Codages  codes = new Codages(tab);
        codes.test(message);
        // FIN TEST CODAGE
    }
}


abstract class Code {

    private String nom;

    public Code(String unNom) {
        nom = unNom;
    }

    public abstract String code(String chaine);
    public abstract String decode(String chaine);

    public void affiche() {
        System.out.print(nom);
    }

    public String getNom() {
        return nom;
    }

}


class ACle extends Code {

    private String cle;

    public ACle(String nomCode, String cle) {
        super(nomCode);
        this.cle = cle;
    }

    public void setCle(String uneCle){
        cle = uneCle;
    }

    public int longueur(){
        return cle.length();
    }

    public String code(String message){
        String codage = "";
        int charMessage;
        int charKey;
        int somme;
        for (int i = 0; i < message.length(); ++i){
            charMessage = message.charAt(i) - 'A' + 1;
            charKey = cle.charAt(i % longueur()) - 'A' + 1;
            somme = (charMessage + charKey) % 26;
            if (somme == 0)
                codage += 'Z';
            else
                codage += (char)('A' + somme - 1);
        }
        return codage;
    }


    public String decode(String codage) {
        String message = "";
        int charCodage;
        int charKey;
        int difference;
        for (int i = 0; i < codage.length(); ++i){
            charCodage= codage.charAt(i) - 'A' + 1;
            charKey = cle.charAt(i % longueur()) - 'A' + 1;
            difference = (charCodage - charKey + 26 ) % 26;
            if (difference == 0)
                message += 'Z';
```

```
            else
                message += (char)('A' + difference - 1);
        }
        return message;
    }


    public void affiche(){
        super.affiche();
        System.out.println(" avec  " + cle + " comme cle");
    }
}

class ACleAleatoire extends ACle
{
    private int length;
    private void genereCle(){
        String newKey = "";
        int randomPosition;
        for (int i = 0; i < length; ++i){
            randomPosition = Utils.randomInt(26);
            newKey += (char)(randomPosition + (int)'A' - 1);
        }
        setCle(newKey);
    }


    public ACleAleatoire(int length){
        super("a cle aleatoire","");
        this.length = length;
        genereCle();
    }

}


class Cesar extends ACle {

    private int crans;

    public Cesar(String nomCode, int crans) {
        super(nomCode, "");
        setCle("" + (char)('A' + crans%26 - 1));
        this.crans = crans;
    }


    public void affiche() {
        System.out.println(getNom() + " a " + crans + " crans");
    }

}


class Codages {

    private Code[] codes;

    private Code cleMax() {
        int max = 0;
        int indexMax = -1;
        for (int i = 0; i < codes.length; ++i){
            if (codes[i] instanceof ACleAleatoire){
                int longueur = ((ACleAleatoire)codes[i]).longueur();
                if (longueur > max){
                    max = longueur;
                    indexMax = i;
                }
            }
        }
        if (indexMax < 0)
            return null;
        else return codes[indexMax];
    }
```

```
    public Codages(Code[] someCodes){
        codes = someCodes;
    }


    public void test(String message){
        String coded;
        for (int i = 0; i < codes.length; ++i){
            System.out.print("Avec le code : ");
            codes[i].affiche();
            System.out.print("Codage de " + message + " : " );
            coded = codes[i].code(message);
            System.out.println(coded);
            System.out.print("Decodage de  " + coded + " : " );
            System.out.println(codes[i].decode(coded));
            System.out.println("-----------------------------------");
            System.out.println();
        }
        Code codeMax = cleMax();
        if (codeMax != null) {
            System.out.println("Code aleatoire a cle maximale :");
            codeMax.affiche();
        }

    }
}
```
