

MOOC Intro POO Java

Exercices semaine 4

Exercice 11 : Affichage et comparaison d'objets

Programmer la hiérarchie de classes "*Rectangle coloré héritant de Rectangle*" en obéissant aux contraintes suivantes :

- La classe `Rectangle` possède les attributs `double` largeur et hauteur.
- La classe `RectangleColore` hérite de `Rectangle` et possède un attribut `couleur` de type `String`
- Le code résultant doit pouvoir être testé avec le programme principal suivant :

```
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.
    }
}
```

et produire alors la sortie :

```
Test 1 :
Rectangle :
  largeur = 12.5
  hauteur = 4.0

Test 2:
Rectangle :
  largeur = 12.5
  hauteur = 4.0
  couleur = rouge

Test 3 :
Rectangle :
  largeur = 12.5
  hauteur = 4.0
  couleur = rouge
true
true
false
false
false
```

- Les méthodes `toString` et `equals` nécessaires ne doivent pas comporter de duplication de code.
-

Exercice 12 : Tour de cartes

Vous vous intéressez dans cet exercice à décrire les données d'un jeu simulant des combats de magiciens.

Dans ce jeu, il existe trois types de cartes : les terrains, les créatures et les sortilèges.

- Les terrains possèdent une couleur (parmi 5 : blanc('B'), bleu('b'), noir('n'), rouge('r') et vert('v').)
- Les créatures possèdent un nom, un nombre de points de dégâts et un nombre de points de vie.
- Les sortilèges possèdent un nom et une explication sous forme de texte.

De plus, chaque carte, indépendamment de son type, possède un coût. Celui d'un terrain est 0.

Dans le programme `Magic.java`, proposez (et implémentez) une hiérarchie de classes permettant de représenter des cartes de différents types.

Chaque classe aura un constructeur permettant de spécifier la/les valeurs de ses attributs. De plus, chaque constructeur devra afficher le type de la carte.

Le programme doit utiliser la conception orientée objet et ne doit pas comporter de duplication de code.

Ajoutez ensuite aux cartes une méthode `afficher()` qui, pour toute carte, affiche son coût et la valeur de ses arguments spécifiques.

Créez de plus une classe `Jeu` pour représenter un jeu de cartes, c'est-à-dire une collection de telles cartes.

Cette classe devra avoir une méthode `piocher` permettant d'ajouter une carte au jeu. On supposera qu'un jeu comporte au plus 10 cartes. Le jeu comportera également une méthode `joue` permettant de jouer une carte. Pour simplifier, on jouera les cartes dans l'ordre où elles sont stockées dans le jeu, et on mettra la carte jouée à `null` dans le jeu de cartes.

Pour finir, dans la méthode `main`, constituez un jeu contenant divers types de cartes et faites afficher le jeu grâce à une méthode `afficher` propre à cette classe.

Par exemple la méthode `main` pourrait ressembler à quelque chose comme cela :

```
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();
    }
}
```

qui produirait quelque chose comme :

```
On change de main
Un nouveau terrain.
Une nouvelle créature.
Un sortilège de plus.
Là, j'ai en stock :
Un terrain bleu
Une créature Golem 4/6
Un sortilège Croissance Gigantesque
Je joue une carte...
La carte jouée est :
Un terrain bleu
```

Exercice 13 : Analyse de programme

Un programmeur amateur produit le code suivant pour tester ses connaissances en POO :

```
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(new Forme(tabFormes[i]));
            formes.dessine();
        }
    }
}
```

```
class Forme
{
    private String couleur;

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

```

Il s'attend à ce que son programme produise l'affichage suivant :

```

Une forme rouge
toute ronde
Une forme jaune
toute pointue

```

Questions:

1. Expliquez pourquoi ce programme principal ne fait pas réellement ce qu'il veut.
2. Qu'affiche-t-il ?
3. Corrigez ce programme de sorte à ce qu'il fasse ce que le programmeur voulait à l'origine. Vous donnerez le code Java de tout élément ajouté en précisant à quelle classe il appartient et indiquerez les éventuelles modifications à apporter à la méthode `main`.

Les contraintes à respecter sont les suivantes :

- la déclaration de `tabFormes` doit rester inchangée;
 - il ne doit pas y avoir de tests de types dans la boucle qui copie les données de `tabFormes` dans la collection.
-

Exercice 14 : Cryptographie

Le but est de créer et implémenter un modèle orienté objet de codes de cryptage. Vous travaillerez dans le fichier fourni `Secret.java`.

```
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;

        // PARTIES A DECOMMENTER AU FUR ET A MESURE SELON l'ENONCE
        /*
        // 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 = { // Décommentez 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
*/
}
}

```

Voici les éléments devant être modélisés :

Code de cryptage

Il s'agit ici d'implémenter une classe `Code`, caractérisant des codes de cryptage.

Un code de cryptage :

- est caractérisé par son *nom*;
- il doit fournir une méthode de cryptage : `String code(String s)`;
- il doit aussi fournir une méthode de décryptage : `String decode(String s)`.

Les méthodes de cryptage et de décryptage dépendent de la nature du code (on en considérera trois, décrits plus loin) et ne peuvent être définies de façon générale.

Il vous est demandé d'implémenter la classe `Code`, de sorte à ce que les contraintes suivantes soient respectées :

1. le constructeur de la classe devra initialiser le *nom* au moyen d'une valeur passée en paramètre;
2. la classe `Code` comportera une méthode `affiche` affichant le nom du code;
3. la classe doit imposer à ses sous-classes de définir les méthodes de cryptage et décryptage;
4. la méthode de cryptage, `code`, prend en argument le message à crypter (une `String`) et retourne le message crypté (une `String` encore);
5. la méthode de décryptage, `decode`, prend en argument le message à décrypter (une `String`) et retourne le message décrypté (aussi une `String`);
6. la classe doit être bien encapsulée.

Dans tout ce qui suit, vous considérerez que les chaînes de caractères à crypter sont constituées uniquement de majuscules et ne contiennent pas d'espaces.

Trois types de codes de cryptage

Concrètement, nous considérerons ici trois types de `Code` :

- les codes à **clé secrète**;
- les codes à **clé secrète aléatoire**;
- les codes à **crans** (dits aussi codes de César).

Il s'agit maintenant d'implémenter ces trois sous-classes de la classe `Code`.

La sous-classe `ACle` :

Un code à clé utilise une autre chaîne de caractères, nommée *clé*, pour encrypter/décrypter les messages. Il fonctionne comme suit :

1. à chaque caractère du message en clair, on associe son rang dans l'alphabet : { 'A' → 1, 'B' → 2, ... 'Z' → 26 }.

Pour le message "**COURAGE**" par exemple :

C	O	U	R	A	G	E
3	15	21	18	1	7	5

La clé est aussi transformée en suite d'entiers, selon le même procédé qu'à l'étape précédente. Pour la clé "**MIX**" par exemple :

M	I	X
13	9	24



2. On aligne l'un au dessous de l'autre le message en chiffres et la clé en chiffres (la clé est répétée autant de fois que nécessaire) :

3	15	21	18	1	7	5
13	9	24	13	9	24	13

3. on additionne en colonne les nombres écrits. Lorsque la somme dépasse 26, on diminue le résultat obtenu de 26. Pour l'exemple précédent :

16	24	19	5	10	5	18
----	----	----	---	----	---	----

4. on remplace la suite de nombres obtenue par les caractères correspondants dans l'alphabet :

P	X	S	E	J	E	R
---	---	---	---	---	---	---

Il vous est demandé de programmer la classe `ACle` caractérisée par un attribut spécifique représentant la clé (une `String`) utilisée pour le codage.

Implémentez cette sous-classe en respectant les contraintes suivantes :

1. le constructeur de la classe initialisera la clé au moyen d'une valeur passée en paramètre;
2. la classe doit être compatible avec la méthode `main` fournie;
3. la classe doit mettre à disposition un méthode `longueur` retournant la longueur de la clé;
4. l'affichage d'un code à clé comprend le nom du code et la clé;
5. la classe doit être proprement encapsulée.

Pour tester cette partie, décommentez la partie du programme principal comprise entre `// TEST A CLE` et `// FIN TEST A CLE`.

La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Avec le code : a cle avec EQUINOXE comme cle
Codage de COURAGEFUYONS : HFPAOVCKZPJWG
Decodage de HFPAOVCKZPJWG : COURAGEFUYONS
-----
```

La sous-classe `ACleAleatoire` :

Implémentez une sous-classe `ACleAleatoire` de la classe `ACle` permettant de générer aléatoirement une clé de longueur particulière.

Cette sous-classe aura donc en plus :

- un attribut indiquant la longueur de la clé souhaitée;
- une méthode `void genereCle()` générant aléatoirement une clé de la bonne longueur et l'affectant à l'attribut `clé` hérité de la super-classe;
- un constructeur compatible avec la méthode `main` fournie (l'argument unique est la longueur de la clé). Ce constructeur fera appel à la méthode `genereCle` et donnera systématiquement "a cle aleatoire" comme nom au code.

La méthode `randomInt` de la classe fournie `Utils` pourra être utilisée pour la génération de nombres aléatoires.

Pour tester cette partie, décommentez la partie du programme principal comprise entre `// TEST A CLE ALEATOIRE` et `// FIN TEST A CLE ALEATOIRE`.

La trace d'exécution devrait ressembler à ce qui suit :

```
Avec le code : a cle aleatoire avec XDBFF comme cle
Codage de COURAGEFUYONS : ASWXGEIHAEMRU
Decodage de ASWXGEIHAEMRU : COURAGEFUYONS
-----
```

La sous-classe `Cesar` :

Les codes à crans, ou codes de César, fonctionnent par décalage.

Chaque lettre du message est remplacée par la lettre apparaissant `n` crans plus loin dans l'alphabet (et ce, de façon cyclique: 'Y' décalé de 4 crans devient 'C').

Par exemple, le cryptage du message "COURAGE" au moyen d'un code à 3 crans produira : "FRXUDJH".

La sous-classe `Cesar` est donc caractérisée par le nombre de crans à utiliser pour le cryptage/décryptage.

Un code à crans, peut-être vu comme une code à clé où :

1. la clé est de longueur 1;
2. le caractère occupant la position *nombre de crans* (modulo 26) dans l'alphabet constitue la clé.

Implémentez cette sous-classe en respectant les contraintes suivantes :

1. le constructeur de la classe initialisera le nombre de crans au moyen d'une valeur passée en paramètre;
2. l'affichage d'un code à crans comprend le nom du code et le nombre de crans;
3. la classe doit être compatible avec la méthode `main` fournie.
4. la classe doit être proprement encapsulée.

Pour tester cette partie, utilisez le programme principal fourni (partie comprise entre `// TEST CESAR` et `// FIN TEST CESAR`).

La trace d'exécution devrait ressembler à ce qui suit :

```
Avec le code : Cesar a 5 crans
Codage de COURAGEFUYONS : HTZWFLJKZDTSX
Decodage de HTZWFLJKZDTSX : COURAGEFUYONS
-----
```

Etude des systèmes de codage

Notre façon de tester les différents systèmes de codage est peu pratique.

Nous souhaitons maintenant rendre ces tests plus simples à réaliser sur un ensemble de `Code`.

Programmez une classe `Codages` caractérisée par un `ensembl` de `Code` à tester.

Cette classe fournira principalement une méthode `void test(String message)` affichant pour chaque code de l'ensemble le résultat du codage de `message` et le résultat du décodage.

Les contraintes à respecter sont les suivantes :

1. la classe `Codages` doit pouvoir être testée avec la méthode `main` fournie;
2. elle fournira une méthode utilitaire `cleMax` retournant le code à clé aléatoire ayant la clé de plus grande longueur. La méthode `test` devra en outre afficher ce code (s'il existe) et n'affichera rien sinon;
3. la classe doit être proprement encapsulée.

Pour tester cette partie, décommentez la partie du programme principal comprise entre `// TEST CODAGE` et `// FIN TEST CODAGE`.

La trace d'exécution devrait ressembler à ce qui suit :

```
Test CODAGES:
-----
```

```
Avec le code : cesar a 5 crans
Codage de COURAGEFUYONS : HTZWFLJKZDTSX
Decodage de HTZWFLJKZDTSX : COURAGEFUYONS
-----
```

```
Avec le code : a cle avec EQUINOXE comme cle
Codage de COURAGEFUYONS : HFPAOVCKZPJWG
Decodage de HFPAOVCKZPJWG : COURAGEFUYONS
-----
```

```
Avec le code : a cle aleatoire avec GUMBL comme cle
Codage de COURAGEFUYONS : JJHTMNZSWKVIF
Decodage de JJHTMNZSWKVIF : COURAGEFUYONS
-----
```

```
Avec le code : a cle aleatoire avec NETYHOZEWL comme cle
Codage de COURAGEFUYONS : QTOQIVEKRKCSM
Decodage de QTOQIVEKRKCSM : COURAGEFUYONS
-----
```

```
Code aleatoire a cle maximale :
a cle aleatoire avec NETYHOZEWL comme cle
```