TD de Compléments en Programmation Orientée Objet n° 1 : Révisions : classes, objets

Exercice 1 : (Re-)prise en main

1. Écrivez une méthode `String[] lireChaines()` qui
   — invite l'utilisateur à saisir 10 chaînes de caractères au clavier ;
   — les stocke dans un tableau ;
   — renvoie ce tableau
2. Questions : que contient le tableau avant que ce programme n'y stocke les chaînes saisies ? Supposons qu'on ait fait cet exercice avec un tableau d'entiers, que contiendrait-il avant d'avoir été rempli ? Quelle est la différence ?
3. Écrivez une autre procédure **void** `enMajuscules(String[] tab)` qui transforme toutes les chaînes du tableau en majuscules (méthode d'objet `toUpperCase()` de `String`).
4. Écrivez une procédure **void** `afficher(String[] tab)`
5. Écrivez un `main()` qui lit les chaînes au clavier, les met dans un tableau, les transforme en majuscule, les trie en ordre alphabétique (méthode statique `Arrays.sort()`), et affiche le résultat.
6. Dans votre code trouvez :
   — les classes ;
   — les objets ;
   — la création ce ces objets ;
   — les appels des méthodes (statiques, non statiques).

Exercice 2 : Personnes

On considère les deux classes suivantes :

```
public class Personne {

    private String nom;
    private String prenom;
    public int age;

    public Personne(String nom, String prenom,
        int age){
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
    }

    public void setPrenom(String p){
        this.prenom = p;
    }

    public void anniversaire(){
        this.age ++;
    }

    public String toString(){
        return "Je m'appelle : " + this.prenom
            + " " + this.nom + ". J'ai "
            + this.age + " ans.";
    }

}

public class Test {

    public static void main(String[] args){
        Personne tony =
            new Personne("Parker","Tony",29);
        System.out.println(tony);
        Personne mickael = tony;
        mickael.setPrenom("Mickael");
        System.out.println(mickael);
    }

}
```

1. Peut-on placer ces deux classes publiques dans un seul fichier ?
2. Qu'obtient-on dans le terminal à l'exécution de Test.**class** ?
3. Peut-on exécuter les lignes suivantes dans le main() pour changer le nom d'une Personne ?

```
mickael.nom = "Gelabale";
System.out.println(mickael);
```

   Sinon, proposez une façon de faire sans modifier les champs de la classe. Étant donné la méthode anniversaire, est-il utile que le champ age soit public ?
4. Si le main avait été écrit dans la classe Personne et non dans la classe Test aurait-on eu le droit d'écrire mickael.nom = "Gelabale"; ?

Exercice 3 : Mot de passe

On souhaite écrire une classe qui corresponde à une entrée de base de données de clients. Une telle entrée est définie par un client (une personne), un login et un mot de passe. Supposons que cette classe aura pour nom Entree.

1. En utilisant la classe Personne définie précédemment, définir la classe Entree et le constructeur adapté.
2. Discuter l'accessibilité des différents champs. Lesquels peuvent-être publics ?
3. Écrire une méthode autorise() qui prend en argument une chaîne de caractères et renvoies un booléen. Elle renvoie **true** si la chaîne fournie correspond au mot de passe. Afin de comparer deux chaînes de caractères, il faut utiliser la méthode equals().
4. Modifier la classe Entree pour afin de stocker le nombre de verifications de mot de passe ratées.
5. Écrire une méthode changerMdp() qui prend en argument deux chaînes de caractères. Si la première correspond au mot de passe, elle remplace le mot de passe actuel par la deuxième.
6. Discuter de l'accessibilité des deux méthodes précédentes.
7. Quels champs de cette classe peuvent être **final** ?

Exercice 4 : Évaluation de code

Qu'affiche le code suivant ? Pourquoi ?

```
class A {
    private int attr;

    A(int value_attr) {
        this.attr = value_attr;
    }

    public bool egal(A b) {
        if(this.attr == b.attr) {
            return true;
        }
        else {
            return false;
        }
    }

    public int getAttr() {
        return this.attr;
    }
}
```

Avec le main() suivant :

```
public static void main(String[] args) {
    A obj = new A(2);
    A obj2 = obj;
    A obj3 = new A(3);

    if(obj.egal(obj2)) {
        System.out.println("Egal");
    }
    else {
        System.out.println("Different");
    }

    if(obj2.egal(obj3)) {
        System.out.println("Egal");
    }
    else {
        System.out.println("Different");
    }

    if(obj.egal(obj3)) {
        System.out.println("Egal");
    }
    else {
        System.out.println("Different");
    }

    if(obj == obj2){
        System.out.println("Egal");
    }
    else{
        System.out.println("Different");
    }

    if(obj == obj3){
        System.out.println("Egal");
    }
    else{
        System.out.println("Different");
    }

    if(obj2 == obj3){
        System.out.println("Egal");
    }
    else{
        System.out.println("Different");
    }
}
```

Expliquez la différence entre l'opérateur == et la fonction egal().

Exercice 5 : Compter

Qu'affiche le programme suivant ?

```
public class Compter {
    private static Element e = new Element(),
        f = new Element();
    public static void main(String [] args)      {
        printall ();
        e.plusUn();
        printall ();
        f.plusUn();
        printall ();
    }
    private static void printall() {
        System.out.println("e : "+print(e));
        System.out.println("f : "+print(f));
    }
    private static String print(Element e)      {
        return "" + e.getA() + e.getB();
    }
}

class Element {
    private static int a = 0;
    private int b = 1;
    public int getA() { return a; }
    public int getB() { return b; }
    public void plusUn() {
        a++; b++;
    }
}
```

Exercice 6 : Transtypage

Dans la méthode main() ci-dessous,

1. Quelles lignes provoquent une erreur de compilation ?
2. Après avoir supprimé ces-dernières, quelles lignes provoquent une exception à l'exécution ?
3. Après les avoir enlevées, elles aussi, quels affichages provoquent les lignes restantes ?

```
class A { }

class B extends A { }

class C extends A { }

public class Tests {

        public static void main(String[] args) {
                System.out.println((int)true);
                System.out.println((int) 'a');
                System.out.println((byte) 'a');
                System.out.println((byte) 257);
                System.out.println((char) 98);
                System.out.println((double) 98);
                System.out.println((char) 98.12);
                System.out.println((long) 98.12);
                System.out.println((boolean) 98.);
                System.out.println((B) new A());
                System.out.println((C) new B());
                System.out.println((A) new C());
        }
}
```

Exercice 7 : Une classe Polygone

Dans cet exercice, nous allons nous pencher sur la modélisation d'une classe pour gérer les polygones réguliers.

1. Quels champs doit contenir la classe Polygone ?
2. Proposez deux méthodes renvoyant l'aire et le périmètre d'un Polygone. *"Rappel :"* l'aire d'un polygone régulier à n côtés le longueur a est $S = \frac{na^2}{4\tan\left(\frac{\pi}{n}\right)}$.
3. Proposez une méthode toString() qui **affiche** le nombre de côtés d'un Polygone.
4. Écrivez une fonction qui affiche un texte définissant les polygones. Cette fonction devrait-elle être **static** ?

Exercice 8 : Listes chainées

Une liste chaînée est une structure de donnée classique pour représenter des listes de données rangées dans un certain ordre. Elle se caractérise par une série de maillons stockés à différents endroits de la mémoire, chaque maillon contenant d'une part une donnée utile et d'autre part une référence vers le maillon suivant.

En Java, une liste chaînée peut être facilement représentée en Java par un objet à deux attributs (le contenu utile, et le maillon suivant). Ainsi la liste vide est égale à **null**, une liste à 1 élément est un maillon dont le deuxième attribut pointe sur la liste vide, et ainsi de suite.

À faire : écrire la classe ListeDEntiers, pour représenter des listes d'entiers, comportant notamment :

1. un constructeur qui prend comme paramètre un entier (contenu du premier maillon) et une liste (référence vers le maillon suivant) ;
2. un constructeur qui prend comme paramètre un tableau d'entiers ;
3. les deux accesseurs (méthodes pour lire les attributs) et "setteurs" (méthodes pour réassigner les attributs)
4. une méthode qui compte et retourne la longueur de la liste ;
5. une méthode qui retourne la valeur du i-ième élément ;
6. une méthode qui supprime le i-ième maillon ;
7. une méthode qui insère un maillon avec une valeur x en position i ;
8. une méthode qui inverse la liste.