Université Paris Diderot **Programmation Orientée Objet**
L2/L3 **Année 2014-2015**

TD - Séance n°1

Révisions – Classes,

> N'oubliez pas de vous inscrire sur DidEL ! Les groupes sont créés, pensez par conséquent à vous iscrire dans *VOTRE* groupe. Si vous n'avez pas le même groupe de TD et TP, inscrivez vous dans les deux.

Lisez attentivement le sujet, les exercices sont toujours plus faciles à faire quand on a *bien* lu l'énoncé. Parfois il peut être utile de lire les questions suivantes : comme ça on sait où l'exercice veut en venir !

Les exercices de la premiere partie du TD doivent tous être traités avant la semaine prochaine. Finissez ceux que vous n'avez pas eu le temps de faire en TD chez vous. Les exercices de la seconde partie sont à faire si vous vous sentez à l'aise.

1 Exercices obligatoires

Exercice 1 *Afficher un objet*

On peut afficher une chaîne de caractères à l'aide de l'appel suivant :

```
String toShow = "hello";
System.out.println(toShow);
```

Affin d'afficher des objets plus compliqués, la classe correspondante doit disposer de la méthode `toString`. Elle ne prend pas d'arguments et renvoie une chaîne de caractères décrivant l'objet. On considère la classe `Point2d` suivante :

```
public class Point2d {

    private double x,y;

    public Point2d(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {
```

```
        // votre code ici
    }

    public static void main(String[] args) {
        Point2d p1 = new Point2d(3.4, 5.6);
        Point2d p2 = new Point2d(8.2, 3.4);
        System.out.println(p1);
        System.out.println(p2);
    }

}
```

Écrivez le code de la méthode `toString` de façon à ce que l'exécution de `Point2d.class` affiche :

```
(3.4; 5.6)
(8.2; 3.4)
```

Exercice 2 *Classes et accessibilité.*

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);
   ```

   Si non, 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 publique ?
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 *Petites manipulations*

1. Modifiez la classe `Personne` de l'exercice précédent, en ajoutant un champ représentant la quantité de monnaie que la personne possède.
2. Les deux méthodes suivantes (redondantes) doivent permettre à deux personnes de se transmettre une certaine somme. La méthode retourne un booléen. Celui-ci vaut `true` si le payement s'est bien déroulé.

   ```
   public static boolean donne (Personne p1,
       Personne p2, int montant) { ... }
   public boolean donne(Personne p,  int montant){ ... }
   ```

   Écrivez ces méthodes. Donnez un exemple d'appel pour chacune.

3. Laquelle de ces deux méthodes vous semble la plus judicieuse ?
4. En commentaire, complétez la spécification en précisant qui paye à qui.

Exercice 4 *Encapsulation*
Dans cet exercice, nous allons définir un compte en banque. Un compte en banque se caractérise par un solde et par un titulaire (en l'occurrence une `personne`).

1. Écrivez une classe Compte, ainsi qu'un constructeur adapte.
2. Écrivez la méthode `getSolde` qui renvoie le montant présent sur un compte.
3. Écrivez la méthode `credite` qui crédite le compte d'un certain montant.
4. Écrivez la méthode `debite` qui débite depuis le compte un certain montant.
5. Écrivez une méthode qui permette au titulaire de retirer une certaine somme pour la mettre dans sa poche. (Pas de découvert)
6. Écrivez une méthode qui permette au titulaire d'effectuer un dépôt, à partir de l'argent qu'il a en poche.
7. *** On se propose d'attribuer un numéro unique à chaque compte. Ainsi, le premier compte instancié aura pour numéro `0`, le suivant `1` et ainsi de suite. En ajoutant à la classe `Compte` un champ statique `nbComptes` et un champ d'objet `numero`, modifiez le constructeur pour qu'il garde trace du nombre de comptes instanciés jusque là et initialise l'identifiant unique `numero`.

2 Si vous avez du temps...

Exercice 5 *Liens croisés*
Dans notre modélisation, un compte est lié à son titulaire, mais une personne ne l'est pas au compte qu'elle possède. Nous proposons ici une modification simple de ce modèle.

1. Modifiez la classe Personne, en ajoutant un champ de type `Compte[]` qui contiendra l'ensemble des comptes associés à une personne.
2. Modifiez le constructeur de `Personne`, en ajoutant un paramètre `int n`. Le constructeur se chargera de fabriquer `n` comptes différents de solde nul pour cette personne.
3. Écrivez dans votre `main` quelques manipulations de crédit sur ces différents comptes.
4. Quand on cherche à retirer une somme importante, il se peut que les fonds d'un seul compte ne suffisent pas. On peut alors vider chacun de nos comptes jusqu'à avoir atteint cette somme. Il se peut également que la somme de tous les fonds ne suffise pas. Écrivez la méthode `retrait` correspondante.

Exercice 6 *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 publiques ?
3. Écrire une méthode `authorise` 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` :

   ```
   String s1 = "haha";
   String s2 = "haha";
   String s3 = "hehe";
   System.out.println(s1.equals(s2)); \\ va afficher true
   System.out.println(s1.equals(s3)); \\ va afficher false
   ```

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` ?