Programmation Orientée Objets

TD/TP n°2 : Classes et Objets

L'objectif de ce TD/TP est de mettre en œuvre les concepts de base de la POO afin de modéliser des personnes. Nous allons créer un paquetage `personnes` (Eclipse autorisé !) composé de deux classes : la classe `Personne` (fichier `Personne.java`) et la classe `Test` (fichier `Test.java`). Cette dernière contiendra une méthode `main`. N'oubliez pas de tester soigneusement chaque exercice.

Exercice 1 (TD) : Diagramme de classes

Dessiner le diagramme de classe(s) modélisant les données du sujet (sans les constructeurs).

Exercice 2 : Variables et constructeur

Écrire une classe `Personne` permettant de modéliser une personne à l'aide des informations (variables) suivantes : Monsieur Pierre Froment, né en 1978, célibataire. Pour représenter certaines informations (ex. Monsieur / Madame), on pourra utiliser des constantes de type `char`.
Ajouter un constructeur à la classe `Personne` permettant d'initialiser le prénom et le nom de la personne. Dans le `main`, déclarer et instancier 3 objets `p1`, `p2` et `p3` de la classe `Personne`.

Exercice 3 : Surcharge de constructeurs

Écrire plusieurs versions du constructeur de `Personne`. Chaque nouvelle version surcharge et appelle la précédente, en initialisant une variable de plus. La dernière version sera utilisée de la manière suivante :

```
Personne p1 = new Personne(Personne.HOMME, "Michel", "Serres",1930);
```

Exercice 4 : Encapsulation

Modifier la classe `Personne` afin de respecter le principe d'encapsulation (visibilité des variables, principaux accesseurs et mutateurs, ex. `String getNom()`). Améliorer la gestion de la date de naissance (jour, mois, année) en utilisant la classe `java.util.GregorianCalendar`.

Exercice 5 : Affichage

Écrire une méthode `String getInfos()` qui retourne une chaîne de caractères identique à celle de l'exemple de l'exercice 2. Vérifier son bon fonctionnement sur les 3 objets `p1`, `p2` et `p3`.
Renommer cette méthode afin de redéfinir la méthode `toString()`, et permettre un transtypage automatique de variables de type `Personne` en chaînes de caractères. Vérifier ce comportement.

Exercice 6 : Âge

Écrire une méthode `int age(int anneeCourante)` qui renvoie l'âge de la personne (nombre d'années) en fonction d'une année donnée en paramètre. Écrire une méthode `boolean plusJeune(Personne p)` qui retourne `true` si la personne courante est strictement plus jeune que la personne `p` donnée en paramètre.
Modifier la méthode `âge` afin d'afficher l'âge en nombre d'années, de mois et de jours.

Exercice 7 : Mariages

Ajouter à la classe `Personne` une variable `conjoint`. Attention : polygamie interdite. Écrire une méthode `void marier(Personne p)` qui permet de marier une personne à une autre. Modifier la méthode `toString()` afin d'afficher le nom des dames « à l'ancienne » :

Mme Brigitte Macron (née Trogneux), née en 1953, mariée.

Exercice 8 : Comptage

Ajouter à la classe `Personne` une constante `NUM_PERSONNE`, initialisée automatiquement à l'instanciation. Écrire et tester la méthode `int getNumero()`.

Exercice 9 : Généalogie

Modifier la classe `Personne` (variables, constructeur) de manière à pouvoir représenter les parents d'une personne (père et mère, ou `null` si l'on ne connait pas le parent). Déclarer et instancier $(2^4 - 1)$ nouvelles personnes sur 4 générations. Écrire une méthode `boolean estAncetre(Personne p)` qui retourne `true` si la personne courante est ancêtre de la personne `p` donnée en paramètre.

Exercice 10 : Arbre généalogique

Écrire une méthode `void afficherArbreGenealogique()` qui affiche l'arbre généalogique d'une personne. Soigner la lisibilité en indentant l'arbre généalogique.