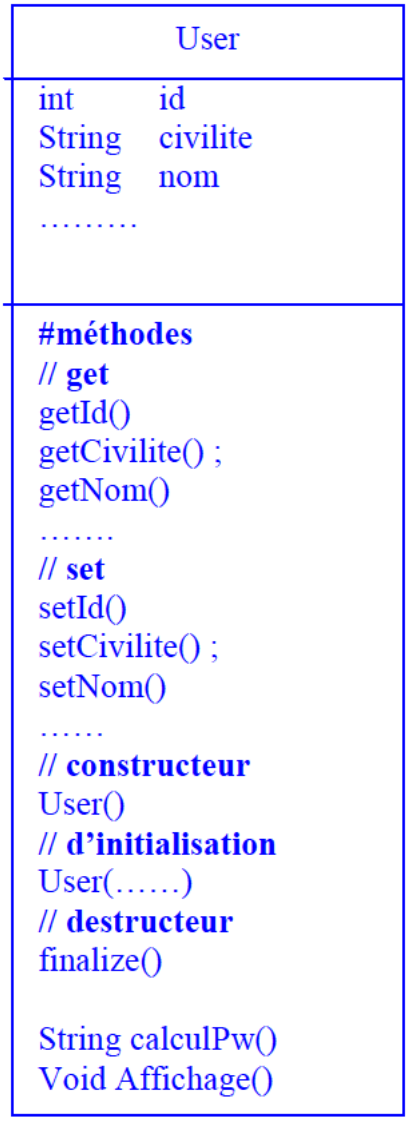
Reprenez Exercice024

Définition d'une classe User avec :

- ses données membres
- ses méthodes d'accès aux données membres
- ses constructeurs et destructeurs



ACCES AUX DONNEES DES OBJETS :

On ne peut accéder directement aux données membres. C'est un problème. Il faut définir à l'intérieur de la classe User des méthodes d'accès à ces données et empêcher l'accès direct aux attributs depuis l'extérieur de la classe.
En général dans les langages Orienté objet ces méthodes se nomme **get** pour la lecture et **set** pour l'écriture.

Exemple : getId() et setId(……..)

Les données membres sont à déclarer en **private** dans la classe. Par ce mot clé, elles ne seront accessibles que dans le corps de la classe, on ne pourra les consulter ou les modifier que si la classe définit une méthode permettant cette consultation ou cette modification.

Par opposition au mot clé **public** où les données membres sont accessibles partout où la classe est accessible.

CONSTRUCTEUR ET DESTRUCTEUR

En Java, le mécanisme d'instanciation attribut par défaut une valeur à chaque variable d'instance en fonction de son type (null pour une variable de type String). Aucune variable d'instance ne peut être indéterminée.
Cette instanciation par défaut convient rarement aux spécifications de la classe. Le programmeur peut définir lui-même l'instanciation qui convient en définissant un ***constructeur***. Un ***constructeur*** est une méthode spéciale, qui spécifie les opérations à effectuer à l'instanciation.

En Java deux types de constructeurs, un par défaut et un d'initialisation.
Le *destructeur* est une méthode particulière qui permet lorsqu'il est activé de retirer (ou nettoyer la mémoire lorsque l'objet n'est plus utilisé).

On utilise la méthode **finalize().**

Aide à la Solution : Voici la gestion de l'encapsulation pour le nom de la classe User

```
// Les attributs doivent être déclarés private
private String nom ;

// lecture ( get )
public String getNom() {
        return nom; // on retourne une chaîne, le nom
}

// écriture ( set )    // on a besoin d'un paramètre : le nouveau nom – pas de return
public void setNom (String newNom) {
        nom = newNom;
}

Pour le test on peut utiliser les différents constructeurs puis on détruit les objets avec finalize( )
public class TP {
        public static void main (String[] args) {
                // 1er user : constructeurs par défaut
                // affichage
                // destruction objet user1

                // 2eme user : constructeurs complet
                User user2;
                user2 = new User(2, "Mr","Martin", .....);

                // le constructeur dans la classe User aura comme forme :

                // public User(int id, String civilite, String nom,…….l) {
                // this.setId(id) ;
                // this.setCivilite(civilite) ;
                // .....
                // }

                // affichage
                // destruction objet user2

                // 3eme user : utilisation des méthodes d'accès aux attributs privés
                User user3 = new User(); // Instanciation
                user3.setNom("DUPONT"); // affectation du nom

                // affichage
                // destruction objet user3
        }
}
```

En plus, en option : Imaginez un constructeur qui réalise la copie d'un objet user passé en paramètre. On veut répondre au message suivant :

```
User user4 = new User(user3) ;
```