

FICHE 2 : ASSOCIATIONS, ENCAPSULATION

Objectifs

- Comprendre et assimiler la notion d'**encapsulation**.
- Être capable d'écrire les "**getter**" et les "**setter**".
- Être capable d'écrire une méthode `toString()`

Vocabulaire

encapsulation	getter	setter	private	public	toString
---------------	--------	--------	---------	--------	----------

Exercices

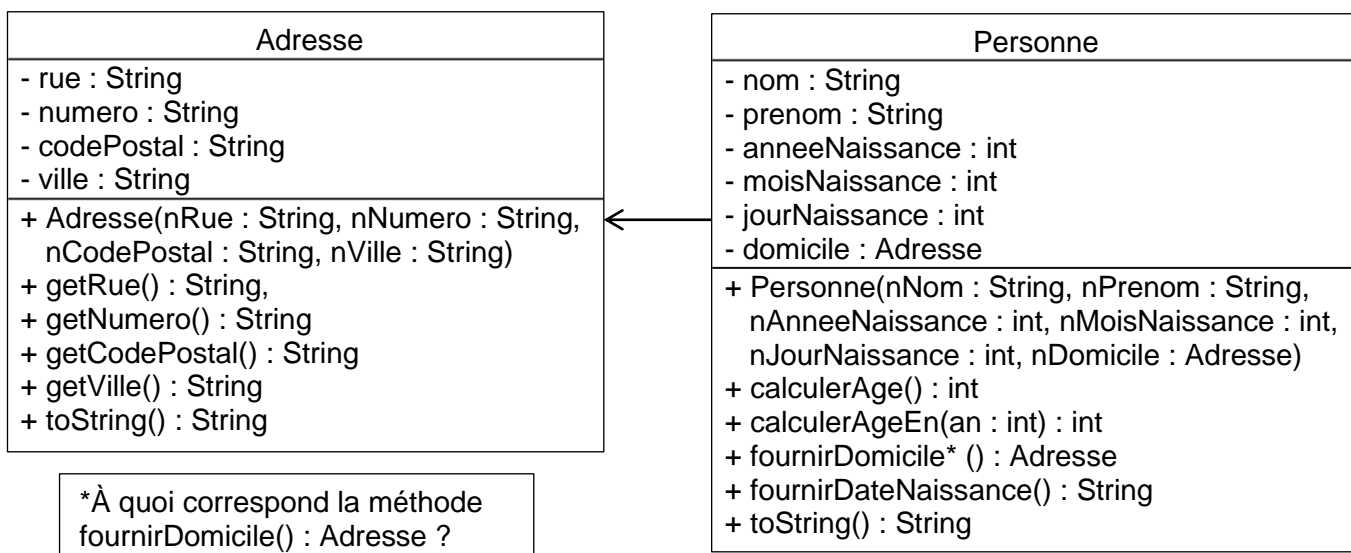
Attention, à partir cette séance, tous les diagrammes de classes ainsi que tous les codes Java des exercices veilleront à l'encapsulation. De manière générale, on dispose d'attributs `private` et de méthodes `public`.

Pour tous les exercices, écrivez toujours une classe `TestNomDeClasseATester` qui validera ce que vous avez écrit.

Créez, dans le répertoire APOO, un projet intitulé **APOO_fiche2** afin d'y implémenter les classes demandées dans les fiches 2, partie 1 et partie 2.

1. Personne – Adresse

Voici les diagrammes des classes `Personne` et `Adresse` :



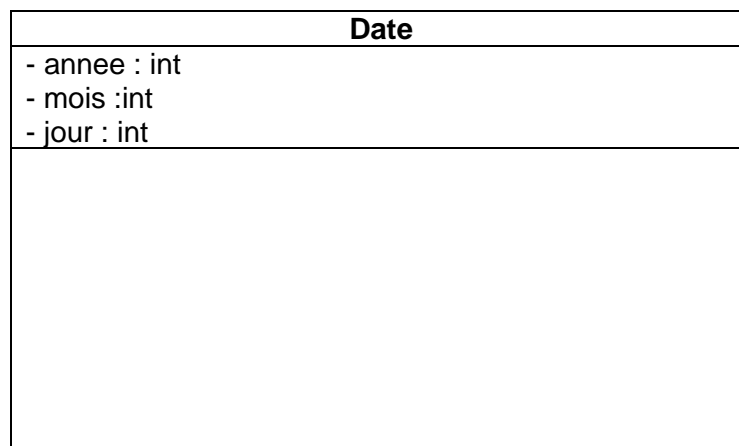
- En vous aidant de la théorie, implémentez ces deux classes en Java.
- Écrivez un programme de test en respectant les étapes précisées ci-dessous :
 - Créer une adresse dans une variable `adresse` avec comme valeurs Rue de la gare n°34 à 5000 Namur.
 - Créer une personne dans une variable `personne1` ayant comme valeurs Paul Schmidt né le 6 février 1968 et dont le domicile est l'adresse ci-dessus.
 - Créer une autre personne dans une variable `personne2` ayant comme valeurs Valérie Gobert né le 7 mars 1970 et dont le domicile est la même que celle de Paul. Ils habitent ensemble.

4. Affichez les informations des deux personnes.

c) Représentez les objets en mémoire à la fin de ce programme :

2. Date

On définit la classe `Date` au moyen du diagramme UML suivant :

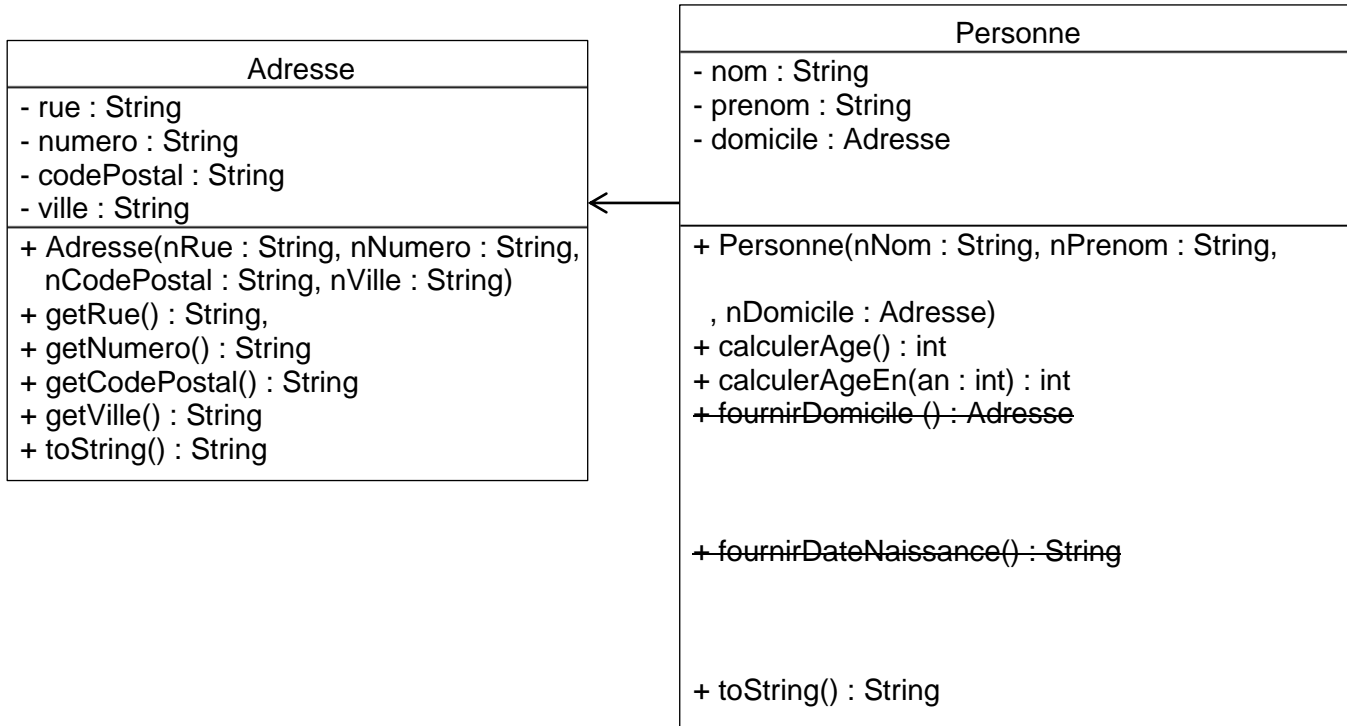


- a) Complétez l'UML de la classe `Date` :
 - ☐ ajoutez un constructeur ;
 - ☐ ajoutez des getters ;
 - ☐ ajoutez une méthode `toString` qui renvoie la date au format «jj/mm/aaaa ».
- b) Implémentez la classe `Date` en Java.
- c) Testez votre classe en créant 2 instances de `Date` et en les affichant.

3. Personne



Reprenez la classe `Personne` et modifiez-la de sorte que la personne conserve un attribut de type `Date` pour sa date de naissance.

- a) Modifiez d'abord votre diagramme UML :
- Ajoutez les getters associés aux attributs `domicile` et `dateDeNaissance`.
 - Adaptez le constructeur de la classe `Personne`.
 - Ajoutez une méthode qui permet de changer l'adresse du domicile. Cette méthode s'intitule `demenager` et prend en paramètres une rue, un numéro, un code postal et une ville. Elle veille à changer le domicile de la personne. Cette méthode ne renvoie rien.



- b) Implémentez ces changements en Java. Vous ne pouvez pas modifier d'autre classe que la classe `Personne`. Il faudra également modifier votre programme de test.

4. Des personnes qui déménagent

- a) Prenez la classe `TestDemenagement` et vérifiez qu'elle s'exécute.
- b) Représentez en mémoire les objets créés dans `TestDemenagement`.
- c) Quel défaut présente l'implémentation de la classe `Personne` telle que fournie ? 
- d) Comment corrigeriez-vous l'implémentation de la classe `Personne` ? 

5. Point

Un point dans un plan est caractérisé par ses coordonnées réelles `x` et `y`. La classe `Point` fournit aussi :

- un constructeur ayant pour paramètres les coordonnées `x`, `y` dans cet ordre ;
- deux getters permettant de récupérer les valeurs de `x` et de `y` ;
- deux setters permettant de modifier les valeurs de `x` et de `y` ;
- une méthode `toString` qui renvoie les coordonnées `x` et `y` au format `(x,y)`.

- a) Donnez l'UML de la classe `Point`
- b) Implémentez cette classe en java et testez-la.