# TP Noté Java

Mathieu Bourgais
mathieu.bourgais@insa-rouen.fr

2023

## Préambule

Vous pouvez utiliser l'éditeur de texte de votre choix. Vous pouvez aussi utiliser un IDE comme Eclipse, Visual Studio Code ou encore Intellij, qui présente de nombreux avantages (auto-complétion, par exemple). Dans Eclipse, vous devez toujours travailler dans un projet (à créer au début de chaque TP/mini-projet) dans lequel vous définirez vos classes.

Pour rappel, pour compiler et exécuter un code source Java, vous pouvez utiliser votre éditeur favori, ou ouvrir un terminal puis naviguer jusqu'au répertoire désiré, et taper :

— javac monFichier.java
— java monFichier

## 1 Exercice 1 - Une première classe simple

### 1.1 Première classe

Créez une classe **Personne**, permettant de représenter une personne à l'aide de son nom et de son prénom. Vous créerez deux constructeurs : l'un ne prenant aucun argument, et initialisant les attributs à la chaîne vide (""), et un autre prenant en paramètre un nom et un prénom, sous forme de chaînes de caractères.

En Java, les attributs d'une classe sont généralement privés. Les autres classes pourront alors lire le contenu de ces attributs à l'aide d'accesseurs **(getters)**, et les modifier à l'aide de mutateurs **(setters)**. Ainsi, pour accéder à l'attribut nom d'une classe, on utilisera l'**accesseur getNom()**. De manière symétrique, pour modifier l'attribut nom, on utilisera le **mutateur setNom(String nom)**.

### 1.2 Accesseurs et mutateurs

Définissez les accesseurs et mutateurs nécessaires à votre classe Personne (pour accéder et modifier les champs "nom" et "prenom").

Compilez votre classe **Personne** ainsi que la classe **TestPersonne** fournie. Cette classe contient la procédure **public static void main(String[] args)** servant de point d'entrée pour l'exécution de votre programme. Corrigez les éventuelles erreurs de compilation liées à votre classe Personne, et vérifiez le bon fonctionnement de votre code.

## 1.3 toString

En Java, toutes les classes héritent de la classe **Object** qui contient notamment la méthode *String toString()* qui permet "d'écrire" un objet (i.e. passer un objet directement dans la méthode System.out.println()).
Néanmoins, cette méthode doit être redéfinie précisément pour chaque classe afin de spécifier comment décrire la classe en question par une chaine de caractères.

Au sein de la classe Personne, redéfinissez la méthode **String toString()**, renvoyant une chaîne de caractères permettant de décrire l'état d'un objet de type Personne, et contenant la valeur de chacun de ses attributs. Vous pouvez, par exemple, décrire une personne en renvoyant une chaîne de caractères de la forme : "Jean Dupont", avec une tabulation entre le prénom et le nom.

Compilez de nouveau votre classe **Personne** ainsi que la classe **TestPersonne** fournie. Quel changement remarquez vous à l'exécution, par rapport à la question 3 ? Que s'est-il passé ?

## 1.4 Héritage

Lors d'un héritage, la classe nouvellement définie est appelée **classe fille**, et le classe dont elle hérite est appelée **classe mère**. La classe fille nouvellement définie **hérite** alors des attributs et des méthodes de la classe mère. En Java, on utilisera pour cela le mot clé **extends**.

Pour rappel, dans les constructeurs de la classe fille, vous devez appeler un des constructeurs de la classe mère à l'aide du mot clé **super**, et des arguments appropriés. Cet appel à super doit toujours être placé en première ligne des constructeurs de la classe fille.

Créez deux nouvelles classes, **Etudiant** et **Professeur**, héritant de la classe **Personne**. Votre classe **Etudiant** ajoutera un attribut représentant la promotion à laquelle l'étudiant est affecté, sous forme d'une chaîne de caractères. La classe Professeur ajoutera un attribut (dont le type est laissé au choix) représentant le salaire mensuel du professeur. Vous ajouterez les constructeurs, accesseurs et mutateurs nécessaires à ces deux nouvelles classes. Vous redéfinirez également la méthode **toString()** dans chacune des classes **Etudiant** et **Professeur**, afin de décrire la valeur de chacun de leurs attributs.

À l'image de la classe **TestPersonne**, créez une classe **TestHeritage** pour tester vos deux nouvelles classes. Vous y instancierez des objets de type **Etu-**

**diant** et **Professeur**, et en afficherez l'état à l'aide des accesseurs et/ou de la méthode **toString()**.

### 1.5 Polymorphisme

Examinez le code de la classe **TestPolymorphisme**. À votre avis, que va-t-il être affiché ? Compilez et exécutez le programme, puis vérifiez votre intuition. Pourquoi le programme se comporte-t-il ainsi ?

## 2 Exercice 2 - Les premières méthodes

Pour cet exercice, vous effectuerez toutes les modifications nécessaires directement au sein des classes **UneClasse** et **UnProgramme** fournies.

### 2.1 Structures conditionnelles

Dans la classe **UneClasse**, écrire une fonction **boolean estPair(int i)** qui prend en entrée un entier i, et retourne True s'il est pair, et False sinon.

Dans la classe **UneClasse**, écrire une procédure **void typeOperation(char c)** qui prend en entrée un caractère c, et affiche "C'est une addition !" si c == '+', "C'est une soustraction !" si c == '-', "C'est une division !" si c == '/', "C'est une multiplication !" si c == '*', et "Quelle drôle d'opération..." sinon. Vous utiliserez l'instruction **switch**.

Compilez votre classe **UneClasse**, puis compilez la classe **UnProgramme**. Que se passe-t-il ? Corrigez les erreurs présentes dans la classe **UnProgramme**, recompilez, et lancez votre programme.

### 2.2 Structures itératives

Les questions suivantes sont à traiter directement dans le corps de la fonction **main** de la classe **UnProgramme**.

À l'aide de la fonction estPair et de l'instruction itérative adéquate, tester la parité de tous les nombres de 0 à 10. Vous afficherez le résultat du test de parité de chaque nombre sous la forme : "0 est pair : vrai", "1 est pair : faux", etc. Pour information, plusieurs chaînes de caractères peuvent facilement être concaténées avec l'opérateur "+" (i.e. System.out.println("une " + "concaténation " + "de plusieurs chaines !") ;)

À l'aide de l'instruction itérative adéquate, calculer et afficher les termes de la suite $U_n$ définie par $u_0 = 1$ et $u_{n+1} = 2 * u_n + 3$, pour tous les n tels que $u_n < 1000$.