# Programmation orientée objets (IG2)
## Classes abstraites et interfaces en Java

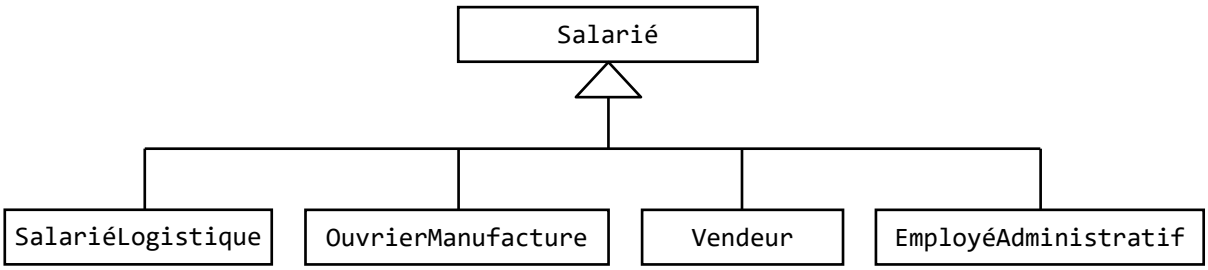
## Exercice 1 : classes abstraites

Une entreprise veut informatiser la gestion de son personnel. Le personnel (l'ensemble des salariés de l'entreprise) sera représenté par un tableau nommé `personnel`. Dans la classe `Principale`, ce tableau sera déclaré et garni pour ensuite calculer le montant total à débourser par l'entreprise pour le paiement de ses salariés.

Les salaires sont calculés de manière différente selon le type des salariés.

- Pour les ouvriers chargés de la logistique, le salaire est calculé en fonction d'un salaire horaire et du nombre d'heures prestées par mois.
- Pour les ouvriers chargés de la manufacture, le salaire est calculé en additionnant une base fixe à un bonus de production dépendant du nombre de pièces produites au cours du mois.
- Pour les employés vendeurs, le salaire est déterminé par une partie fixe et une partie variable calculée sur base du total des ventes effectuées au cours du mois (CA).
- Pour les employés administratifs, le salaire mensuel est fixe.

On utilisera donc la hiérarchie de classes et d'interfaces suivantes :




### Étape 1

Dans la fonction principale de la classe `Principale`, on désire pouvoir utiliser le code suivant :

```
Salarié [] personnel = new Salarié [5];
// Garnir le tableau personnel
double totalSalaire = 0;
for (int i = 0 ; i < personnel.length ; i++) {
  System.out.println(personnel[i].getNom() + " : " + personnel[i].getSalaire());
  totalSalaire += personnel[i].getSalaire();
}
System.out.println("Salaire total : " + totalSalaire);
```

Cela signifie que la classe `Salarié` doit avoir une méthode `getNom()` donnant le nom du salarié et une méthode `getSalaire()` indiquant le salaire d'un salarié. Cependant, il est impossible de définir cette dernière méthode à ce niveau-là : en effet, le calcul du salaire s'effectue de manière différente pour chacun des types des salariés. Il devra donc s'agir d'une **méthode abstraite**.

Construisez la classe abstraite `Salarié` avec un attribut privé (`nom`) et deux méthodes. Ajoutez-lui un constructeur. *Question : pourquoi définir un constructeur alors qu'il est interdit d'utiliser «* `new Salarié(…)` *» vu qu'il s'agit d'une classe abstraite ?*

## ■ Étape 2

Définissez la classe `EmployéAdministratif` descendant de `Salarié`. Un employé administratif est décrit par son nom et par son salaire mensuel. Implémentez la méthode `getSalaire()`.

Modifiez la méthode principale pour réduire la taille du tableau `personnel` à 1. Garnissez la première (et unique) case de ce tableau avec le salarié suivant : Hubert, employé administratif, gagnant 1500 € par mois.

**Note.** En plus du « `for` » habituel, le Java permet d'utiliser un format appelé « `for each` » et qui permet de passer en revue tous les éléments d'un tableau (mais aussi d'une liste, d'un arbre, etc.). Voici une version équivalente de la méthode principale.

```
Salarié [] personnel = new Salarié [5];
// Garnir le tableau personnel
double totalSalaire = 0;
for (Salarié salarié : personnel) {
  System.out.println(salarié.getNom()  + " : " + salarié.getSalaire());
  TotalSalaire += salarié.getSalaire();
}
System.out.println("Salaire total : " + totalSalaire);
```

## ■ Étape 3

Définissez la classe `Vendeur` descendant de `Salarié`. Un vendeur est décrit par son nom, la région où il opère, son salaire fixe et le montant du chiffre d'affaires réalisé le mois précédent. Son salaire est égal au fixe augmenté de 10% de son chiffre d'affaires.

Modifiez la méthode principale pour augmenter la taille du tableau `personnel` à 2. Ajoutez le salarié suivant dans la 2e case : Julie, vendeuse sur la région de Namur, 500 €, 7500 € de chiffre d'affaires.

## ■ Étape 4

Définissez la classe `SalariéLogistique` pour représenter les ouvriers de la logistique qui sont caractérisés par un nom, un salaire horaire et un nombre d'heures prestées. Le salaire d'un ouvrier de la logistique se calcule en multipliant son salaire horaire par le nombre d'heures prestées.

Agrandissez le tableau `personnel` et ajoutez-y Gérard, ouvrier de la logistique, qui a presté 115 heures de travail rémunérées à 10,5 € de l'heure.

## ■ Étape 5

Définissez la classe `OuvrierManufacture` pour représenter les ouvriers de la manufacture. Chacun de ces ouvriers est caractérisé par son nom, le type d'objets qu'il fabrique, l'objectif mensuel à atteindre en termes de nombre d'objets à fabriquer, le nombre d'objets qu'il a effectivement fabriqués au cours du mois, son salaire de base et la prime qu'il peut décrocher s'il a atteint son objectif.

Agrandissez le tableau `personnel` et ajoutez-y les deux ouvriers suivants.

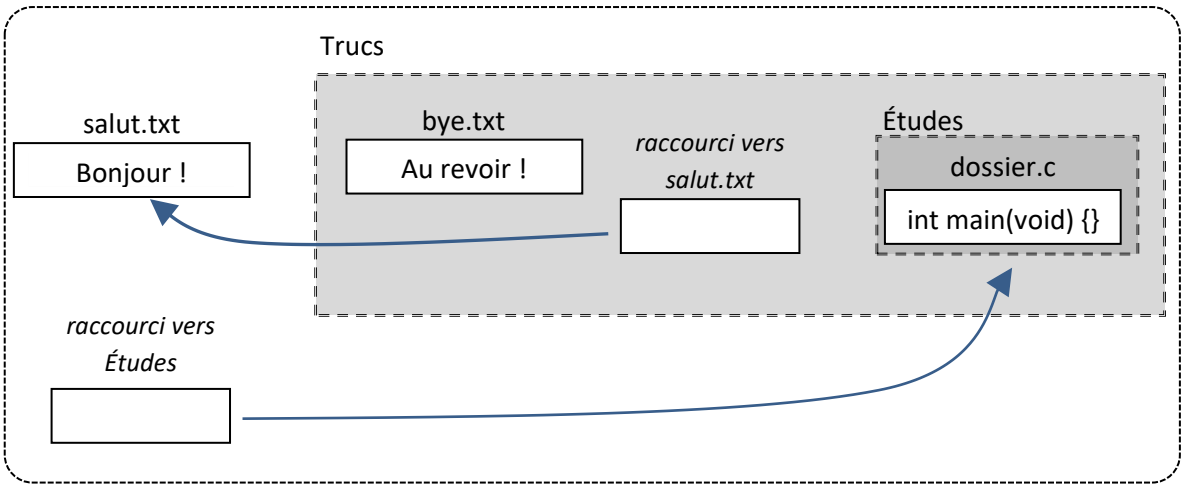
- Marc, qui fabrique des "bougies", avec un salaire de base de 1100 € et une prime de 150 €. L'objectif à atteindre est de 2500 bougies par mois. Le mois passé, Marc en a fabriqué 2700 (il reçoit donc la prime).
- Luc, qui fabrique des "roues", avec un salaire de base de 1150 € et une prime de 100 €. L'objectif à atteindre est de 225 roues par mois. Le mois passé, Luc en a fabriqué 210 (il ne reçoit donc pas la prime).

## Exercice 2 : interfaces

Dans cet exercice, on va programmer un petit bout d'un operating system (comme Windows, Linux et autres) et, plus particulièrement, la gestion du contenu d'un disque dur.

L'operating system en question permet d'avoir trois types « d'éléments » : des fichiers, des répertoires et des raccourcis (au sens de « raccourcis vers un fichier / un répertoire / un autre raccourci », comme sur Windows). Pour chacun de ces éléments, il doit être possible de récupérer son nom et de réaliser l'action associée (en double-cliquant sur l'élément en question).

Le but de cet exercice est de pouvoir représenter le contenu d'un disque dur comme, par exemple, celui qui est représenté sur la figure ci-dessous.




- La figure indique que le disque dur contient
  - un fichier appelé « salut.txt » et contenant le texte « Bonjour ! » ;
  - un raccourci vers le répertoire « Études » ;
  - un répertoire appelé « Trucs » et comportant les éléments suivants :
    - un fichier appelé « bye.txt » et contenant le texte « Au revoir ! » ;
    - un raccourci vers le fichier « salut.txt » ;
    - un répertoire appelé « Études » et comportant un seul fichier :
      - le fichier appelé « dossier.c » et contenant le texte « int main(void) {} ».

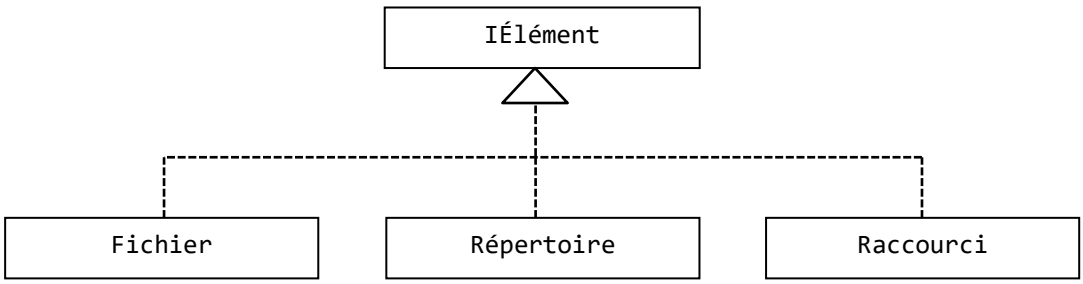
Le tableau suivant décrit les particularités des trois types d'éléments.

| | **Caractérisé par…** | **Son nom est…** | **Action associée** |
|---|---|---|---|
| **Fichier** | • un label<br>• une extension<br>• un contenu (String) | = label suivi d'un point puis de l'extension | Afficher son contenu |
| **Répertoire** | • un nom<br>• un contenu (qui est un tableau d'éléments) | donné directement | Afficher le nom des éléments que le répertoire contient |
| **Raccourci** | • une cible (qui est un autre élément) | = « raccourci vers » suivi du nom de la cible | L'action de la cible |

Pour les fichiers et les répertoires, le nom est donné. Les raccourcis, par contre, ont un nom formé de « raccourci vers » et du nom de leur cible. (Un raccourci vers un raccourci vers salut.txt s'appellera donc « raccourci vers raccourci vers salut.txt ».)

Quand on double-clique sur un fichier, il faut afficher son contenu (en pratique, c'est un peu plus compliqué que ça). Si on double-clique sur un raccourci, tout se passe comme si on avait double-cliqué sur la cible de ce raccourci. Si on double-clique sur un répertoire, il faut lister les éléments qu'il comporte : un dossier peut comporter des fichiers et des raccourcis mais également d'autres répertoires (qui, eux-mêmes, auront un contenu). Le contenu d'un répertoire est donc un tableau d'éléments.

On représente la situation en Java en utilisant la hiérarchie suivante.




Chaque élément doit pouvoir donner son nom (méthode `String getNom()`) et effectuer une action en cas de double clic (`void action()`). Ces deux méthodes se trouveront donc dans la classe « `Élément` ». Cependant, à ce niveau-ci, aucune d'entre elles ne peut être implémentée, car la manière dont on obtient le nom d'un élément et l'action associée dépendent du type d'élément. Il s'agira donc de méthodes abstraites… et, comme toutes les méthodes sont abstraites, on préférera définir une interface `IÉlément` (par convention, on fait commencer le nom des interfaces par I).

## ■ Étape 1

Définissez l'interface `IÉlément` comme indiqué ci-dessus. L'interface `IÉlément` décrit le « cahier des charges » que devra remplir chaque élément.

## ■ Étape 2

Définissez la classe `Fichier`, un type d'élément. Un fichier est caractérisé par un nom, une extension et un contenu (tous les trois des chaînes de caractères). Implémentez un constructeur ainsi que les

deux méthodes. Dans la méthode principale, définissez les trois fichiers de l'exemple (salut.txt, bye.txt et dossier.c) et testez les méthodes.

## ■ Étape 3

Définissez la classe `Répertoire`, un second type d'élément. Un répertoire est caractérisé par un nom et par un tableau d'éléments (donc, un tableau de `IÉléments` !). Dans le cadre de cet exercice, on ne va pas gérer l'ajout et la suppression d'un élément dans un dossier. On pourra donc se contenter d'un constructeur recevant le nom du répertoire et un tableau d'éléments déjà garni. Dans la méthode principale, définissez le répertoire Études de l'exemple et testez les méthodes.

## ■ Étape 4

Définissez finalement la classe `Raccourci`, un troisième type d'éléments. Un raccourci est caractérisé par une cible, qui est un autre élément (donc, un `IÉlément`). Implémentez un constructeur et les deux méthodes. Dans la méthode principale, définissez les deux raccourcis de l'exemple (celui qui cible un fichier et celui qui cible un répertoire) et testez les méthodes.