Harmonisation/Propédeutique POO

Anne-Marie Pinna-Dery & Mireille Blay-Fornarino
Version originale par Anne-Marie Pinna-Dery

Certains éléments des leçons ont été rédigés avec l'aide de chatGPT.

Prérequis : Vous savez tous écrire des fonctions et/ou des procédures qui prennent des arguments et renvoient des valeurs.
Objectifs : Acquérir les premiers principes de la POO que vous approfondirez en cours et les illustrer avec le langage Java, support technique du cours de POO.
Planning :

Les 2 premières séances ont pour objectif de vous présenter les bases de la POO avec la notion de classes et d'instances.

Un test sera fait en début de séance 3 pour vérifier le niveau d'acquisition des concepts de base avant de poursuivre.

Les 2 séances suivantes seront consacrées davantage à la modélisation objet en mettant l'accent sur la création d'une application avec plusieurs classes.

Table des Matières

Étude de cas : Modélisation du fonctionnement de l'harmonisation

Tout au long de votre formation, vous serez amené à résoudre des études de cas, ce qui implique d'aborder des problèmes concrets présentés par des clients. Vous devrez analyser ces cas pour comprendre les besoins, puis proposer et concevoir des solutions qui seront mises en œuvre en utilisant un langage de programmation. Au cours de cette harmonisation, nous suivrons la méthode suivante : nous aborderons ensemble une étude de cas spécifique, en la développant étape par étape.

Étude de cas :
Dans le cadre de la spécialité informatique de Polytech Nice, les premières semaines sont consacrées à une période préliminaire d'introduction aux cours principaux de la première année nommée *Harmonisation*.
Pendant cette période, des enseignants de l'école sont responsables de cours d'introduction qui sont délivrés aux étudiants inscrits en fonction de leurs connaissances préalables dans la matière.
Les principales données importantes sont les suivantes :

1. Un cours est caractérisé par son intitulé (une chaine de caractères), les étudiants inscrits, un niveau de difficulté noté entre 1 (facile) et 5 (difficile) et un enseignant responsable.
2. Plusieurs cours sont proposés (POO, Réseaux, …) aux étudiants entrants en fonction de leur formation préalable mais l'accès reste ouvert aux étudiants qui le souhaitent sous réserve qu'ils s'inscrivent au cours.
3. Chaque étudiant est défini par son nom, prénom et son année de naissance.
4. Un enseignant est également décrit par son nom et prénom mais aussi par le numéro de son bureau par exemple A 321. Les enseignants qui n'ont pas de bureau ont un numéro par défaut qui est « A 0»
5. D'autres informations peuvent être ajoutées pour gérer au mieux les semaines d'introduction comme une note d'auto-évaluation correspondant à la progression de l'étudiant pendant les semaines.

Ces informations vont nous permettre par un programme de :

1. Obtenir la liste des étudiants inscrits à un cours
2. Obtenir la moyenne d'âge des étudiants d'un cours
3. Obtenir la liste des cours suivis par un étudiant
4. Obtenir le nombre et la moyenne des niveaux de difficulté des cours suivis par un étudiant.

Il est également important de proposer une solution fiable qui permet de travailler avec des informations correctes et cohérentes sur cette période en caractérisant la liste des enseignants impliqués, des cours et des étudiants afin de par exemple

5. Obtenir la liste des cours proposés en Harmonisation
6. Vérifiez qu'un étudiant inscrit à un cours est bien référencé par ce cours
7. Vérifiez que tout enseignant a au moins un cours dont il est responsable

Nous pouvons facilement voir que ce type de problème relativement précis pourrait être repensé ou étendu pour proposer des outils de gestion d'année Polytech.

PARTIE 1- Modéliser une classe simple et l'implémenter en Java

I. La notion de Classe et de variables d'instance

Leçon

Face à un nouveau problème, on réfléchit en identifiant les principaux concepts manipulés dans ce programme. Pour cela on va avoir une réflexion que l'on qualifie ***d'orientée objet***. On ne manipule plus que des **objets** qui contiennent des données et qui savent répondre à des questions en exécutant un code très proche de ce que vous savez faire quand vous écrivez des fonctions ou des procédures. Chaque concept correspond à une *classe*.

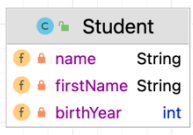
Illustration avec la notion d'*étudiant :*
« Chaque étudiant est défini par un nom, prénom et son année de naissance »

L'étudiant est un "concept important à modéliser" car nous allons devoir la manipuler dans le programme.
Chaque classe a plusieurs **instances** : il y a potentiellement autant d'instances différentes d'étudiants que d'étudiants inscrits en SI3.
Quand on décrit une classe, on peut préciser les données qui permettent de distinguer une instance d'une autre. On appelle ces données des **variables d'instances**.
Par exemple pour un étudiant, il y a au moins 3 variables d'instances :

- le nom (une chaîne de caractères, par exemple "Haddock"),
- un prénom (une chaîne de caractères, par exemple "Archibald"),
- son année de naissance (un entier par exemple 2002)

Le code suivant vous présente une première définition possible de cette classe, la classe **Student**, avec sa visualisation dite UML à gauche. Les variables d'instances *name* et *firstName* sont « private » c'est-à-dire qu'elles ne sont accessibles qu'à l'intérieur de la classe. La classe est définie dans **un fichier** qui porte son nom, ici ***Student.java***

Notez que le nom d'une classe commence par une majuscule en java.
Notez que le nom des variables d'instances commence par une minuscule et si c'est un nom composé, on utilise une majuscule pour distinguer les 2 mots (firstName).




```java
public class Student {
    private String name;
    private int birthYear;
}
```

Questions en groupe

1. Quelles sont les variables d'instance de la classe Student ?
2. Que représentent-elles ? Quel est leur type ?
3. Quelle variable a été oubliée ? Quel est son type ?


Le mot clé *private* permet de protéger les données qui ne seront visibles que dans la classe. On appelle cela **le principe d'encapsulation.**

Question individuelle & Codage

Un **enseignant** est décrit par un nom, un prénom et la référence à son bureau par exemple A 321.

4. Créer la classe **Teacher** (donc bien dans un nouveau fichier qui s'appelle Teacher.java) et définir ses variables d'instances.

II. Constructeurs (V0)

Leçon

Une instance peut être vue comme une boite noire (une structure en mémoire) qui contient ses données.

Pour créer des instances à partir d'une classe (créer les espaces mémoire contenant les données), il faut définir des **constructeurs**.

En l'absence de constructeur explicite, Il existe un constructeur par défaut qui alloue l'espace mémoire, mais avec des valeurs initiales des données dépendantes du type, voire inexistante. Pour pouvoir initialiser une instance à sa création, il vaut mieux implémenter ses propres constructeurs, par exemple Student(String aName, String firstName, int aBirthYear) est un constructeur qui nous permet de créer un étudiant en précisant son nom, son prénom et son année de naissance. Dans le code ci-dessous «this» désigne le nouvel objet.

```
public Student(String aName, String firstName, int aBirthYear) {
    name = aName;
    this.firstName = firstName;
}
```


public Student(String aName, String firstName, int aBirthYear) est la **signature** du constructeur qui correspond **au nom de la classe** suivi d'autant de paramètres que nécessaire pour initialiser les variables d'instances. La suite entre { et } est le **corps de la méthode**, ici un constructeur, ensemble des instructions qui vont s'exécuter à l'appel du constructeur.

Questions en groupe & Codage individuel

5. Que fait ce constructeur ? Nous avons volontairement défini deux affectations des variables, pourquoi utilise-t-on *this* à la 2e et pas à la 1e ?
6. Quelle instruction a été oubliée ?

Question individuelle & Codage

7. On souhaite pouvoir créer un Enseignant à partir de son nom et son prénom, uniquement.

Ajouter le constructeur correspondant dans la classe **Teacher**

III. Méthodes d'instances, focus sur les accesseurs

Leçon

Quand on décrit une classe, on peut préciser les comportements des instances. On appelle l'expression de ces comportements, **méthodes d'instances**. On dit qu'on invoque une méthode sur une instance lorsqu'on demande à une instance d'exécuter le corps de la méthode à partir de ses propres données.
Une **instance** doit être vue comme une boite noire (une structure en mémoire) qui contient ses données. Pour pouvoir accéder aux données en lecture (resp. en écriture) de l'extérieur de la classe, il faut utiliser une méthode spécifique qu'on appelle **accesseur en lecture,** par exemple String **get**Name(), (resp **en écriture ou mutator,** par exemple void setName(String name)).
On peut également définir une méthode d'instances qui va calculer l'âge d'un étudiant par rapport à une année donnée. Les codes suivants présentent les différentes implémentations associées.

Remarquez l'utilisation de *this* dans setName pour bien distinguer la variable d'instance email de l'objet (*this.name*) du paramétre *name*.

La convention de nommage veut que les accesseurs en lecture aient un nom préfixé par **get** et les accesseurs en écriture par **set.**

void est utilisé quand la méthode ne renvoie pas de valeur (c'est une procédure et non une fonction).
return désigne la valeur renvoyée par une fonction. Bien sûr, la valeur doit être du même type que le type dans la signature de la méthode. Par exemple, dans getName(), email doit être de type **String ou la signature de l'accesseur est incorrecte.**

Attributs et méthodes sont définis dans leur classe qui commence par exemple par { et se termine par }

public class Student {

//variables, méthodes et constructeurs sans ordre prédéfini et se termine par }

```
public String getName() {
    return name;
}

public int getBirthYear() {
    return birthYear;
}

public void setName(String name) {
    this.name = name;
}

public void setBirthYear(int birthYear) {
    this.birthYear = birthYear;
}
```

On définit également la méthode suivante :

```
public int ageIn(int year) {
    return year - birthYear;
}
```

Questions en groupe & Codage individuel

8. Quelles sont les variables d'instances utilisées dans ce code ? Quels sont les paramètres ?
9. Ajouter les accesseurs en lecture et écriture au prénom.
10. Que fait la méthode *ageIn* ? que prend-t-elle en paramètre ?

Question individuelle & Codage

11. On souhaite pouvoir accéder en lecture au *nom* de l'enseignant, à son *prénom* et au nom du bureau.

12. On souhaite pouvoir associer à un enseignant un nom de bureau sur la base suivante : *nom du bâtiment* et *numéro*

```
Par exemple, office = building + " " + number;
```

Le + permet de concaténer des chaines de caractères. Notons qu'ici number est un entier et que l'opérateur + fait une conversion automatique en chaine de caractères.

13. Modifier la classe Teacher pour correspondre à cette spécification.
    Pour vous guider, combien de paramètres sont nécessaires pour enregistrer le nouveau nom d'un bureau ?

IV. Instances et appels aux constructeurs

Leçon

En appliquant le premier constructeur on peut créer une instance « s » d'un étudiant en précisant son nom, son prénom et son année de naissance :

```
Student s = new Student("Doe", "Jane", 2000);
```

Il est possible d'obtenir son nom : s.getName(), de le stocker dans une variable par exemple String sonNom = s.getName() etc.

Le **.** permet **d'envoyer un message** à une instance pour demander d'exécuter une méthode publique de sa classe.

Questions en groupe & Codage individuel

14. Étudiez le code suivant qui se trouve dans la classe *Student*
    i. Que fait le code en ligne 46 ?

System.out.println vous permet d'afficher la chaîne en paramètre sur votre écran.

    ii. Que fait le code en ligne 48 et 49
    iii. Comparez ces deux précédentes lignes de code

```
44    public static void main(String[] args) {
45        Student s = new Student( herName: "Doe",  firstName: "Jane",  dateOfBirth: 2000);
46        System.out.println(s.getName());
47        System.out.println(s.getFirstName());
48        int age = s.ageIn( year: 2021);
49        System.out.println(age);
50        System.out.println(s.ageIn( year: 2021));
51
```

La **méthode *main*** est une méthode spécifique qui permet d'exécuter du code décrit dans les classes.

15. Ajoutez le code précédent à votre classe *Student*
    public static void main(String[] args) { …

16. Exécutez le code et vérifier la cohérence des résultats obtenus.
17. Définissez une nouvelle instance de Student qui a pour nom « Doe » pour prénom « John » et est née en 2001.
18. Vérifiez vos résultats. Plus tard, vous apprendrez à tester vos codes. Ici, nous "vérifions" simplement en regardant que les codes se comportent bien comme nous l'attendons.
19. Que se passe-t-il si on exécute les instructions suivantes :

```
Student s = new Student("Doe", "Jane", 2000);
System.out.println(s);
System.out.println(s.getName());
s.setName("Haddock");
System.out.println(s.getName());
```

20. Que se passe-t-il si on exécute les instructions suivantes :

```
Student s1 = new Student("Doe", "John", 2000);
Student s2 = new Student("Doe", "Jane", 2000);
s1 = s2;
System.out.println(s1.getFirstName());
s1.setName("Schmitt");
System.out.println(s2.getName());
```

21. Que se passe-t-il si on demande l'exécution des instructions suivantes :
    a) s.setName(Dupo) ;
    b) s.getName(12) ;
    c) s.getNom();

Question individuelle & Codage

22. Ajoutez un « main » dans votre classe **Teacher** et créez des instances de Teacher
23. Vérifiez que vous pouvez bien accéder au nom de l'enseignant.
24. Faites appel à la méthode qui permet de modifier le nom de son bureau et vérifier le résultat obtenu.

V. Visibilités

Leçon

Vous avez dû remarquer la présence des mots clefs **public** et **private**.

Par exemple la déclaration de la classe *Student* est publique, c'est-à-dire que n'importe quelle partie du programme peut y accéder. Nous avons fait de même pour toutes les méthodes que nous avons écrites. Inversement toutes les variables d'instances sont déclarées comme privées dans nos codes. On ne peut y accéder qu'au travers des méthodes publiques définies par la classe, y compris les accesseurs. Il est vivement recommandé de toujours protéger ainsi le contenu de vos instances, on appelle cela **le principe d'encapsulation.**

Questions en groupe & Codage individuel

25. Étudiez le code suivant
   a) Qu'est-ce ? Dans quelle classe est-il défini ?
   b) Que fait le code en ligne 11 ? (Ce code est équivalent à this.isValid…)
   c) Que remarquez-vous sur la méthode en ligne 18 ? Pouvez-vous expliquer pourquoi elle est privée ?
   d) Que fait-elle ?
   e) Que fait le code des lignes 11 à 16 ?
   f) Identifiez la présence des returns et des booleans.
   g) Quelle valeur est affectée lorsque l'année de naissance est considérée comme erronée ?
26. Implémentez ce code, adaptez-le pour que les dates soient plus réalistes (avoir plus de 15 ans aujourd'hui semblerait une bonne limite 😊) et vérifiez qu'il fonctionne.

```
8     public Student(String aName, String firstName, int dateOfBirth) {
9         name = aName;
10        this.firstName = firstName;
11        if (isValidBirthYear(dateOfBirth)) {
12            this.birthYear = dateOfBirth;
13        }
14        else {
15            this.birthYear = 1901;
16        }
17    }
      1 usage  new *
18    private boolean isValidBirthYear(int dateOfBirth) {
19        return (dateOfBirth >= 1900) && (dateOfBirth <= 2021);
20    }
```

Question individuelle & Codage

27. Modifiez le code de la méthode suivante pour ne modifier le nom du bureau que si le numéro est supérieur à 0 et est strictement inférieur à 500.

```
public void buildOfficeName(String building, int number) {
        this.office = building + " " + number;
    }
}
```

28. On considère à présent que nom du bâtiment doit être une lettre comprise entre A et F. Ces contraintes pourront évoluer et on ne souhaite pas qu'il soit possible d'y accéder en dehors de la classe. Proposez une implémentation d'une méthode privée qui vérifie que les informations données sont correctes avant de modifier le nom du bureau.

Le code suivant est là pour vous aider.
L'appel à la méthode length() retourne la taille de la chaine de caractère.
L'appel à la méthode charAt() retourne le caractère qui se trouve en position 0 dans la chaine de caractères.
Attention un caractère tel que 'A' est noté entre simples côtes alors qu'une chaine de caractères comme "Doe" est notée entre double côtes.

```
if ((building.length() != 1) || (building.charAt(0) < 'A') ||
(building.charAt(0) > 'F')) {
    return false;
}
```

 length et charAt sont des méthodes publiques de la classe String prédéfinies dans Java.

VI. Visibilité en action

Leçon

Nous poursuivons sur la visibilité des variables d'instances « privée ».
Si nous ajoutons le code suivant à la classe *Student*, il compile parce que dans la classe qui les définit les propriétés, variables ou méthodes, même privées sont accessibles.

```
public static void main(String[] args) {
…
Student s1 = new Student("Doe", "Jonh", 2000);
System.out.println(s1.name + " " + s1.firstName + " agée de " +
s1.ageIn(2023));
if (s.isValidBirthYear(2023))
    s.setBirthYear(2023);
else System.out.println("too young!!");

}
```

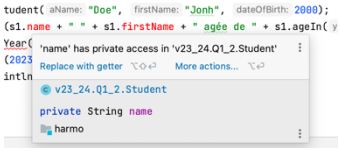
Nous définissons à présent une nouvelle classe ainsi :

```java
public class TestStudents {
    new *
    public static void main(String[] args) {

        Student s = new Student( aName: "Doe", firstName: "Jane", dateOfBirth: 2000);
        System.out.println(s);
        System.out.println(s.getName());
        s.setName("Haddock");
        System.out.println(s.getName());

        Student s1 = new Student( aName: "Doe", firstName: "Jonh", dateOfBirth: 2000);
        System.out.println(s1.name + " " + s1.firstName + " agée de " + s1.ageIn( year: 2023));
        if (s.isValidBirthYear( dateOfBirth: 2023))
            s.setBirthYear(2023);
        else System.out.println("too young!!");
    }
```

Les lignes 13 et 14 sont en erreur. En effet, il n'est pas possible d'accéder aux variables d'instance privée de la classe *Student* depuis la classe *TestStudents* et de même pour la méthode *isValidBirthYear*.



Questions en groupe & Codage individuel

29. Implémentez ces codes, vérifiez et comprenez. Donc créez une classe *TestStudents*
30. Comment pouvez-vous quand même accéder aux variables d'instance depuis la classe *TestStudents*?
31. Pensez-vous qu'il est logique de tester la valeur de l'année en dehors de la classe *Student* elle-même ? Modifier le code de la classe Student pour n'affecter une nouvelle date de naissance que si elle respecte les contraintes énoncées.

Question individuelle & Codage

32. Créez la classe *TeachersTests* et vérifiez que le *main* fonctionne.

VII. Cascades de constructeurs

Leçon

Dans le contexte de la création de plusieurs constructeurs au sein d'une classe en Java, une pratique courante consiste à les faire s'appeler. Cette approche permet d'éviter la duplication de code, tout en simplifiant la création d'objets par le biais de diverses combinaisons d'arguments. De manière concrète, par exemple, un constructeur plus général définit l'initialisation des variables. Un autre constructeur, moins spécifique l'invoque avec des valeurs par défaut. Cette démarche en cascade favorise non seulement la clarté du code, mais aussi sa maintenabilité.

1) Nous souhaitons pouvoir créer des étudiants uniquement avec un nom et un prénom et mettre comme année de naissance dans ce cas 1901.

Nous définissons donc un nouveau constructeur ainsi :

```
public Student(String aName, String firstName) {
    this(aName, firstName, 1901);
}
```

Ce nouveau constructeur qui permet de créer une instance de *Student* sans connaitre son année de naissance et en faisant appel par **this(…) en première ligne du constructeur** au constructeur que vous avez défini précédemment.

2) Nous introduisons une nouvelle variable d'instance dans la classe Student

```
private String email;
```

On ne souhaite pas pouvoir modifier son contenu depuis l'extérieur mais seulement le créer automatiquement à la création de l'instance comme la conjonction du nom et du prénom et de l'adresse de l'université.
On ajoute donc uniquement un accesseur en lecture et on modifie le constructeur pour que l'adresser email soit construite à partir du nom et du prénom et de l'adresse de l'université.

Questions en groupe & Codage individuel

33. Introduisez une nouvelle variable d'instance dans la classe *Student*

```
private String email;
```

34. On ne souhaite pas pouvoir modifier son contenu depuis l'extérieur mais seulement l'affecter automatiquement à la création de l'instance comme la conjonction du nom et du prénom et de l'adresse de l'université.
    a. Ajoutez donc uniquement un accesseur en lecture
    b. Modifiez la construction de l'objet pour affecter la variable *email* par

```
this.email = name + "." + firstName + "@etu.univ-cotedazur.fr";
```

    c. Quel constructeur choisissez-vous de modifier et pourquoi ?

35. Actuellement l'adresse *email* est donc construite à la construction de l'objet et la valeur obtenue est stockée dans la variable d'instance *email*.
    a) Mais que se passe-t-il si vous modifiez le nom de l'étudiant ?
    b) Modifiez votre accesseur en écriture sur le nom, pour que toute modification du nom, respectivement du prénom, mettre à jour la valeur de la variable d'instance représentant *l'email*.
    c) Avez-vous du code dupliqué ? Quelles améliorations proposez-vous pour éviter la duplication de code ?
    d) Une autre solution aurait été de définir, comme nous l'avons fait pour l'âge une unique méthode qui construit *l'email* à la demande. Nous choisissons ici de garder la variable d'instance *email*.
    e) Quels sont d'après vous les avantages et inconvénients de chaque solution ?

Question individuelle & Codage

36. Ajoutez la variable d'instance *email* à Teacher dans les mêmes conditions que précédemment, sauf qu'elle se termine par : "@univ-cotedazur.fr"

37. Mettez à jour le constructeur de la classe *Teacher* pour qu'il affecte par défaut le nom du bureau à « A 0 » et tienne compte la nouvelle variable d'instance email.

38. Ajoutez un constructeur qui prend en paramètre le nom, le prénom et le nom du bureau.

VIII. toString

Leçon

Avez-vous remarqué ce qui est affiché si vous exécutez le code suivant ?

```
Student s = new Student("Doe", "Jane", 2000);
System.out.println(s);
```

Vous devez avoir un affichage ressemblant à :
`Student@1f32e575`, ce qui n'est pas très utile.

Par défaut, lorsque l'on demande à afficher un objet, celui s'affiche sous la forme du nom de la classe et une référence mémoire.
Mais il est possible de définir la méthode **toString** dans chaque classe, pour afficher des informations spécifiques à l'objet. La méthode **toString** en Java permet de convertir un objet en une représentation sous forme de chaîne de caractères. Cela facilite l'affichage et le débogage en fournissant une description significative de l'objet. Par défaut, la méthode renvoie le nom de la classe et une référence mémoire, mais elle peut être redéfinie pour afficher des informations spécifiques à l'objet.

Par exemple, soit la méthode suivante définie dans la classe Student

```
public String toString() {
    return "Student : \n\t" + name + "\t " + firstName + ", \n\t" + email +
", \n\t" + birthYear + "\n";
}
```

Nous obtiendrons à présent comme affichage :

Student :
Doe Jane,
Doe.Jane@etu.univ-cotedazur,
2000

Notez que
• \n permet d'aller à la ligne et \t d'ajouter une tabulation
• +avec une variable de type int fait automatiquement la conversion de type de int à String.
• System.out.println appelle automatiquement la méthode toString()
• \' permet d'ajouter un ' dans une String par exemple
"name='" + name + '\'' correspond à name='Doe'

Questions en groupe & Codage individuel

39. Ajoutez la méthode *toString* à la classe *Student*
40. Testez-la par des appels à System.out.println avec en paramètre des instances de Student.
41. Adaptez l'affichage à votre convenance.

Question individuelle & Codage

42. Définissez la méthode toString de la classe Teacher.
43. Voici une proposition d'affichage, mais vous pouvez choisir celle qui vous convient :

Teacher: name='Doe',firstName='John',email='Doe.John@univ-cotedazur.fr'., office='A 0'

IX. Constantes

Leçon

Avez-vous remarqué qu'à plusieurs reprises nous avons utilisé une même valeur comme par exemple « A 0 » pour donner un nom par défaut à un bureau, 1901 comme date de naissance par défaut ?
Il est préférable lorsque l'on manipule de telles valeurs qui ne seront jamais modifiées de les définir comme des constantes.

```
static final String DEFAULT_OFFICE = "A 0";
```

En ajoutant **static** à la déclaration de la variable nous exprimons qu'elle sera partagée par toutes les instances de la classe Teacher.
En ajoutant **final** à la déclaration de la variable nous exprimons qu'elle ne pourra pas être modifiée dans le code.

```
public Teacher(String name, String firstName) {
    this.name = name;
    this.firstName = firstName;
    this.office = DEFAULT_OFFICE;
    buildEmail();
}
```

Questions en groupe & Codage individuel

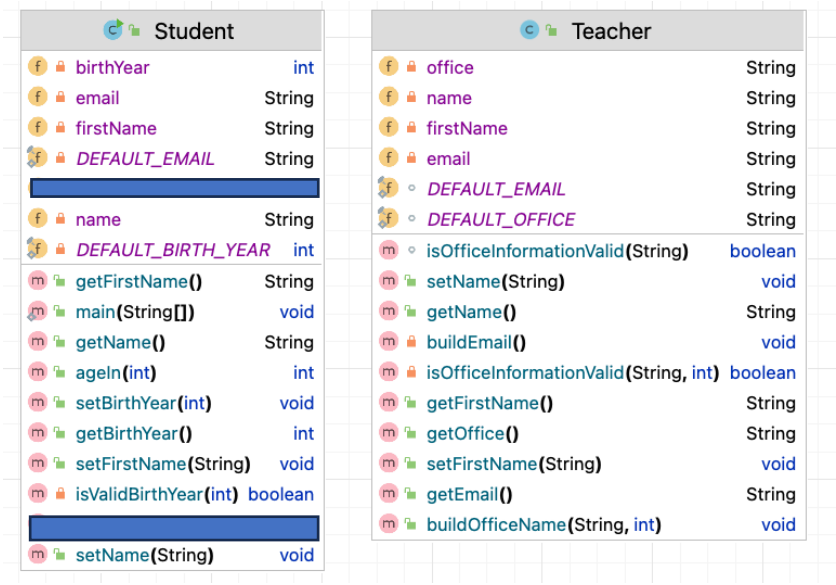
44. Mettez à jour la classe Teacher avec la déclaration de cette nouvelle constante et remplacer dans tout votre code les occurrences de « A 0 »
45. Ajouter une constante dans la classe *Student* pour représenter l'année de naissance par défaut dont la valeur est 1901 et remplacer toutes les occurrences de 1901.
46. Quelle autre constante pourrait être définie ?

Question individuelle & Codage

47. Vérifiez que vos codes s'exécutent « bien ».

X. Bilan

Vous devriez avoir une classe *Student* et une classe Teacher définies par les éléments suivants.



Nous avons ajouté la méthode suivante pour vérifier dans le constructeur de Teacher que le nom du bureau est valide.

```
boolean isOfficeInformationValid(String officeName) {
    if (officeName.length() <3) {
        return false;
    }
    if ((officeName.charAt(0) < 'A') || (officeName.charAt(0) > 'F')) {
        return false;
    }
    if (officeName.charAt(1) != ' ') {
        return false;
    }

    int number = 0;
    String numberPart = officeName.substring(2,officeName.length() - 1);
    try {
        number = Integer.parseInt(numberPart);
    }
    catch (NumberFormatException e) {
        return false;
    }
    return (number >= 0) || (number <= 500);
}
```

Questions en groupe & Codage individuel

48. Comprenez le code de la méthode qui a été ajoutée et ajoutez-le à votre code en vérifiant dans le constructeur adéquat que le nom du bureau est bien formé.