Développement Orienté Objets

IUT Montpellier-Sète – Département Informatique

- **Support de cours**
- **Enseignants:** Marin Bougeret, Gaëlle Hisler, Victor Poupet, Gilles Trombettoni, Petru Valicov
- Le forum Piazza de ce cours pour poser vos questions
- Email pour toute question concernant le cours.
- Le sujet du TP en format .pdf téléchargeable et imprimable.

TP 3 : *Java* - références, constructeurs, première application orientée objet

On se propose ici de réaliser une application de gestion des étudiants dans un département informatique d'un IUT. L'objectif est de développer l'application de manière incrémentale en ajoutant les fonctionnalités demandées au fur et à mesure.

Il est vivement recommandé d'utiliser au maximum les fonctionnalités de l'IDE pour réaliser les tâches courantes (renommage d'attributs/méthodes, génération des différentes méthodes : constructeurs, setters, getters, etc.).

Important : Afin de garder une trace de la progression de votre application, il vous est demandé de travailler dans **un package différent pour chaque exercice**. Pour cela, vous copierez les classes écrites pour un exercice dans le package de l'exercice suivant à l'aide de l'IDE dans le panneau *Project* à gauche pour qu'il corrige automatiquement les déclarations de package. Garder une trace de progression pour chaque exercice vous permettra de mieux comparer votre travail pour chaque exercice et vous permettra également de mieux *réviser* plus tard.

Consignes

- Sauf indication contraire, **tous** les attributs que vous allez déclarer dans ce TP (et dans les TPs qui suivent) doivent être privés (`private`).
- A priori, la plupart des méthodes devraient être déclarées publiques (`public`). Vous pouvez tout de même déclarer et utiliser des méthodes `private` du moment qu'elles vous sont utiles et que votre programme fonctionne correctement.
- Date limite de rendu de votre code sur le dépôt GitHub : **dimanche 13 février à 23h**

Exercice 1

1. Créez et implémentez la classe `Etudiant` avec les attributs suivants : nom, prénom, date de naissance, adresse mail, adresse postale. Pour représenter les dates vous pouvez utiliser la classe statique `LocalDate` du package `java.time`. Voici un exemple de création d'une date :

   ```
   import java.time.LocalDate;

   class GestionEtudiants {
       public static void main(String[] args) {
           LocalDate maDate = LocalDate.of(2021, Month.FEBRUARY, 4);
       }
   }
   ```

   Astuce : Si vous utilisez la classe `LocalDate` dans votre code sans l'importer, l'IDE vous proposera automatiquement de l'importer (placez le curseur sur le nom de la classe et appuyez sur Alt+↵ pour ajouter automatiquement la ligne `import java.time.LocalDate;` au début de votre fichier).

   Pour représenter les autres attributs vous pouvez utiliser le type `String`. Munissez également la classe d'un constructeur.

Astuce : L'IDE peut générer automatiquement le code des méthodes usuelles pour une classe. Ainsi, après avoir déclaré les attributs de votre classe, faites un clic droit dans le code de la classe et sélectionnez «*Generate...*» (ou Alt+Insert) puis choisissez «*Constructor*». Sélectionnez les attributs que vous souhaitez passer directement en argument au constructeur (ici tous) et validez. Vous pouvez évidemment modifier par la suite le constructeur ainsi généré.

2. Créez la méthode `toString()` (qui renvoie un objet de type `String`) générant un texte de présentation des informations concernant l'objet `Etudiant`.

   Attention : cette méthode n'affiche rien, elle se contente de produire une chaîne de caractères. Ce à quoi sert ce texte (par ex. à être affiché à l'écran, ou être écrit dans un fichier) dépend de l'utilisation de la méthode.

3. Ajoutez une fonction `setNom(...)` qui permet de changer le nom d'un objet de type `Etudiant`.

4. Vérifiez votre solution dans le programme principal (la classe `GestionEtudiants`) en instanciant dans la méthode `main(...)` une variable `lolo` de type `Etudiant` et en affichant ses informations.

   Astuce : L'IDE possède des raccourcis pour les morceaux de codes fréquemment utilisés. Par exemple, pour faire un affichage à la console avec `System.out.println()`, tapez «sout» à l'endroit où vous souhaitez insérer l'instruction la fonction et appuyez sur la touche ↹ (tabulation) pour que «sout» soit remplacé par la déclaration complète (`System.out.println()`). Vous pouvez voir la liste des raccourcis disponibles à tout moment en appuyant sur Ctrl+J (les raccourcis disponibles dépendent du contexte, par exemple si vous êtes en train d'écrire une méthode, directement à la racine d'une classe, etc.).

5. Créez une nouvelle variable `toto` de type `Etudiant` construite avec exactement les mêmes paramètres que `lolo`. Comparez les deux variables avec l'opérateur ==. Que constatez-vous ?

6. On se rend compte qu'en fait, `toto` est juste un surnom de `lolo`. Modifiez votre programme pour faire en sorte que `toto` fasse référence à `lolo`. En invoquant la méthode `setNom(...)` sur l'objet `toto`, vérifiez que la modification se répercute bien sur `lolo`.

7. Créez une classe `Departement` qui aura comme attributs une spécialité, une adresse et un tableau d'étudiants inscrits géré sous forme de liste. Pour déclarer une telle liste vous pouvez utiliser la classe `ArrayList` du package `java.util` (voir un exemple dans le cours). Munissez la classe d'un constructeur, qui prend comme paramètre un intitulé et une adresse. Définissez la méthode `toString()` dans `Departement` avec un texte qui liste l'ensemble des étudiants du département.

   Indication : Pensez à utiliser les fonctionnalités de l'IDE pour importer la classe `ArrayList` et pour générer le constructeur et la méthode `toString()`.

8. Ajoutez une méthode `inscrire(...)` dans la classe `Departement` qui prend en paramètre un étudiant et l'ajoute aux étudiants inscrits du département.

9. Ajoutez une méthode `desinscrire(...)` qui supprime un étudiant passé en paramètre de la liste des étudiants inscrits.

10. Simulez votre application dans la classe principale en créant un département `monDepInfo` et en y inscrivant quatre étudiants (dont `toto` et `lolo` définis précédemment). Désinscrivez ensuite `toto` du département. Que constatez-vous ?

Exercice 2

Rappel : Pour préserver le code écrit dans l'exercice précédent, copiez l'ensemble des classes du package `fr.umontpellier.iut.exo1` dans le package `fr.umontpellier.iut.exo2` en utilisant les outils de *refactoring* de l'IDE. Pour ce faire : clic droit sur le nom de la classe → *Refactor* → *Copy*

On souhaite étoffer le modèle objet conçu auparavant en y incluant les aspects pédagogiques du département. Pour cela on vous demande de gérer les *matières* et les *notes*.

1. Une `Matiere` est définie par un intitulé, un coefficient (une valeur réelle qui pourra servir pour le calcul d'une moyenne). Créez la classe correspondante avec un constructeur adéquat. Ajoutez dans `Matiere` une méthode accesseur `getCoefficient()` qui retourne le coefficient.

2. Une `Note` est définie par une matière et par un nombre réel (qui représente la valeur de la note). Déclarez la classe correspondante et ajoutez des méthodes accesseurs pour chacun des attributs.

3. Ajoutez à la classe `Etudiant` un attribut correspondant à la liste de ses notes et une méthode `noter(...)` qui prend en paramètre une matière et une valeur réelle, crée un objet de type `Note` et l'ajoute à la liste des notes de l'étudiant.

4. Sans toucher au code des autres classes, ajoutez à la classe `Etudiant` une méthode `calculerMoyenne()` qui permet de calculer la moyenne pondérée des notes de l'étudiant. Pensez à ajouter des tests unitaires pour vérifier le bon fonctionnement de cette fonction. Pour créer une classe de tests unitaires, placez-vous dans la classe que vous souhaitez tester et :

- appuyez sur Alt+Insert (ou bien faites un clic droit sur le nom de la classe → *Generate*)
- Choisissez *Test...*
- dans l'onglet *Testing library* vous choisirez l'option *JUnit 5*
- donnez un nom approprié à votre classe de tests unitaires (par ex. `EtudiantTest`) et cliquez sur *Ok*.
- Comme pour le TP précédent, la classe de tests générée sera automatiquement placée dans le même package que la classe testée et dans le répertoire correspondant aux tests. Pour écrire vos tests, vous pouvez vous inspirer des exemples vues dans le TP précédent. Vous devez ajouter au moins les tests suivants :

  ```
  test_calcul_moyenne_retourne_zero_Quand_Pas_De_Note()
  test_calcul_moyenne_retourne_valeur_note_quand_une_seule_note()
  test_calcul_moyenne_retourne_valeur_note_pondérée_quand_une_seule_note_avec_coefficient()
  test_calcul_moyenne_retourne_moyenne_quand_plusieurs_notes_avec_differents_coefficients()
  ```

  Indication : Il est assez difficile de comparer des nombres réels, car dans presque tous les langages de programmation, ils sont représentés en virgule flottante et par conséquent, ils sont *approximés* (plus de documentation pour Java sur le site d'Oracle). Une façon simple de comparer des réels et de les comparer à un epsilon près :

  ```
  assertEquals(double expected, double actual, double delta)
  ```

5. Vérifiez que votre programme fonctionne bien en l'exécutant depuis la classe principale.
6. Construisez le diagramme de classes correspondant au programme *Java* que vous avez écrit.

Exercice 3 (Bonus)

Observez que la classe `Etudiant` a un constructeur avec 5 paramètres. Bien entendu, le nombre de paramètres risque de croître car beaucoup d'autres attributs sont susceptibles d'être ajoutés à la classe `Etudiant`. Observez aussi que lorsqu'on construit une instance d'`Etudiant` il est facile de se tromper dans l'ordre des paramètres du constructeur. Heureusement que l'IDE vous aide en vous suggérant cet ordre lorsque vous programmez... De plus, les valeurs de certains attributs peuvent être inconnues au moment de la construction de l'objet : l'adresse de l'étudiant n'est pas encore connue car l'attribution des logements universitaires par les organismes correspondants n'a pas encore eu lieu, l'adresse mail n'est pas encore active au moment de l'inscription de l'étudiant, etc. Plusieurs solutions peuvent être envisagées :

1. Une solution est de définir plusieurs constructeurs avec différents paramètres et de les faire collaborer (voir exemples en cours). Ainsi, l'utilisateur pourra choisir le constructeur qui lui convient. C'est ce qu'on appelle une *construction télescopique*.

   Créez une nouvelle classe `EtudiantTelescopique` (en copiant les attributs de la classe `Etudiant` de l'exercice 2) et modifiez-la afin de pouvoir instancier les étudiants de différentes manières :

   - en indiquant uniquement le nom et le prénom
   - en indiquant uniquement le nom, le prénom et la date de naissance
   - en indiquant uniquement le nom, le prénom et le mail

   Vérifiez votre programme dans la classe principale.

2. Avec cette approche, est-il possible d'ajouter un constructeur supplémentaire afin d'indiquer uniquement le nom, le prénom et l'adresse ? Pourquoi ?

3. Une autre solution serait de suivre le modèle *Java Beans*, en proposant un seul constructeur minimal et en fournissant des méthodes modifieurs (*setters*) pour chaque attribut de la classe. Créez une telle classe appelée `EtudiantJavaBeans`. Voici comment se fera alors la construction des objets de ce type :

   ```
   class GestionEtudiants {
       public static void main(String[] args) {
           EtudiantJavaBeans toto = new EtudiantJavaBeans();
           toto.setNom("Dupont");
           toto.setDateDeNaissance(LocalDate.of(2003, Month.JANUARY, 28));
           /* ... */
           toto.setAdresse("99, av. Occitanie, 34000 Montpellier");
       }
   }
   ```

4. Comparez cette solution avec celle de la classe `EtudiantTelescopique`. Quels sont les avantages et les inconvénients ?

5. Finalement, on vous demande de développer une solution en combinant les bonnes idées des deux versions précédentes. Voici comment on voudrait pouvoir créer des étudiants dans la classe principale :

```
class GestionEtudiants {
    public static void main(String[] args) {
        Etudiant lolo = new EtudiantBuilder()
            .ajouterNom("Dupont")
            .ajouterPrenom("Philippe")
            .ajouterDateNaissance(LocalDate.of(2003, Month.JANUARY, 28))
            .ajouterMail("dupont@etu.umontpellier.fr")
            .ajouterAdresse("99, av. Occitanie, 34000 Montpellier")
            .build();
    }
}
```

   Remarquez qu'il n'y a qu'un seul symbole `;` dans le programme ci-dessus. Autrement dit, l'instanciation de l'objet se fait avec une seule instruction, qui a été écrite sur plusieurs lignes pour plus de lisibilité.

   Ajoutez la classe `EtudiantBuilder` à votre application pour que l'instruction ci-dessus fonctionne.

Pour aller plus loin : Vous noterez que la classe `EtudiantBuilder` sert uniquement à instancier des objets `Etudiant` de manière "organisée" et lisible. Le problème est qu'il est toujours possible d'instancier un objet de type `Etudiant` sans utiliser le *builder* que vous avez écrit... C'est pour cela qu'il est possible d'améliorer la dernière solution en déclarant la classe `EtudiantBuilder` comme classe interne statique de la classe `Etudiant` et de rendre `private` le constructeur de la classe `Etudiant`. Ainsi la construction pourra se faire exclusivement à travers le *builder* et celui-ci servira uniquement à la construction des objets de type `Etudiant`, car c'est son seul rôle. Pour plus de détails et explications, voir le modèle de conception *Builder*. Une explication approfondie est donnée dans *Effective Java*, J. Blosch, (2nd or 3rd edition).