Septembre 24

Harmonisation/Propédeutique POO

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

Cette version est dédiée aux étudiants qui ont déjà programmé en Java ou C++
L'objectif est d'aligner leur vocabulaire et leur pratique de la POO avec les principes enseignés en POO en SI3

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.

Outil d'aide à la gestion de l'Harmonisation

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 sur lesquelles vous devez vous baser pour la suite 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. On souhaite pouvoir calculer l'âge d'un étudiant par rapport à une année donnée.
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. Vérifier qu'un étudiant inscrit à un cours est bien référencé par ce cours ;
7. Vérifier 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'années à Polytech.

PARTIE 1- Ecrire les classes Student et Teacher

Informations complémentaires

1. Contraintes sur la référence à un bureau : le numéro doit être supérieur à 0 et strictement inférieur à 500 et la lettre doit être comprise entre A et F.
2. Contraintes sur l'email : défini comme la conjonction du nom et du prénom et de l'adresse de l'université. Il ne faut pas permettre de modifier le contenu de l'email hors des classes dans lequel il est défini.

Bonnes pratiques à suivre

1. Définir les constructeurs utiles comme Student(String aName, String firstName, int aBirthYear) qui permet de créer un étudiant en précisant son nom, son prénom et son année de naissance.
2. Définir les accesseurs en lecture et en écriture (mutators) utiles.
3. Définir une classe de Test contenant une **méthode *main*** pour exécuter et tester vos codes.
4. Vérifier la cohérence des informations passées en paramètre comme l'année de naissance.
5. Définir la méthode **toString** dans chaque classe, pour former une chaine de caractères contenant les informations spécifiques à l'objet.
6. Utiliser le polymorphisme et l'héritage à bon escient afin de faciliter la compréhension et la réutilisation du code écrit.

PARTIE 2 – Relations entre classes : classes Course et Harmo

Conservez les bonnes pratiques ci-dessus.

Informations complémentaires :

1.Représenter l'ensemble des cours délivrés en harmonisation par un ensemble (Set et HashSet).

2. Un enseignant connait la liste des cours dont il est responsable. Évidemment un cours dont il est responsable doit avoir pour responsable lui-même.

Implémenter cette spécification en tenant bien compte de la bi-directionnalité entre le responsable de cours et la liste des cours dont un enseignant est responsable : un enseignant choisit à un cours c1 dont il prend la responsabilité.

Besoins supplémentaires

Complétez le code pour pouvoir

1. Pouvoir faire des recherches dans Course afin de
    a. Savoir si un Etudiant est inscrit à un cours ou pas.
7. Retrouver un étudiant à partir de son nom uniquement.
8. Retrouver plusieurs étudiants à partir d'un nom.
9. Retrouver plusieurs étudiants à partir d'une année de naissance.
10. Faciliter la recherche de cours dans Harmo
11. Vérifier si un enseignant est bien défini comme enseignnat du cours qu'il gère
2. Obtenir la position (rang d'inscription) d'un étudiant de nom donné dans un cours.
3. Obtenir la moyenne du niveau de difficulté des cours choisis par un étudiant.
4. Obtenir la méthode qui calcule la moyenne du niveau de difficulté des cours dont il est responsable.
5. Obtenir la moyenne d'âge des étudiants de l'harmonisation.

Supposons que nous voulions ajouter le fait que les étudiants évaluent leur progression à la fin du cours sur les notions abordées avec un pourcentage. Comment proposez- vous de définir cette donnée et de faire évoluer le code en conséquence ?

AMUSEZ-VOUS avec vos codes !

PARTIE 3 – Contrôlez vos connaissances

Vocabulaire à maîtriser

Différences entre Classes, Instances ou objets.
Différences entre variables d'instances, attribut, field, constantes, paramètres
Différences entre méthodes, méthodes d'instances, constructeurs
Différnces entre Fonction et procédure
Différences entre accesseurs en lecture et mutators
Différences entre signature et corp d'une méthode

Principe d'encapsulation
Différence entre public protected et private

Principe de nommage
Contraintes de nommage pour le nom : d'une classe, d'un attribut, d'une méthode, d'une constante, d'un accesseur en lecture / écriture

Questions

1. Quelles sont les variables d'instance de la classe Student ?
2. Que représentent-elles ? Quel est leur type ?
3. A quoi sert le mot-clé private. Quand l'utilise-t-on ?
4. Que se passe-t-il en l'absence de constructeur explicite,
5. Qu'appelle-t-on des constructeurs en cascade ?
6. Quand utilise-t-on void ? return ?
7. Que désigne le mot-clé «this» quand il est associé à une variable d'instances ?
8. Que désigne le mot-clé «this» quand il est utilisé comme instruction dans un constructeur ?

Rappels

1. Le + permet de concaténer des chaines de caractères. Notons si un des opérandes est un entier et l'autre une chaine, l'opérateur + fait une conversion automatique en chaine de caractères.
2. Le . permet d'envoyer un message à une instance (par exemple, etudiant instance de Student) pour demander d'exécuter une méthode publique définie dans sa classe (par exemple getName()).
3. Les String commencent à l'indice 0 comme tous les tableaux en java.
4. L'appel à la méthode length() retourne la taille de la chaine de caractères.
5. L'appel à la méthode charAt() retourne le caractère qui se trouve en position en paramètre (ici 0) dans la chaine de caractères.
6. length et charAt sont des méthodes publiques de la classe String prédéfinies dans Java.
7. Attention un caractère tel que 'A' est noté entre simples côtes alors qu'une chaine de caractères comme "Doe" est notée entre double côtes.
8. != signifie différent.

9. || exprime un ou. L'opérande à droite n'est évalué que si celle de gauche est fausse, donc dans l'exemple ci-dessous que si la longueur est égale à 1.

10. \n permet d'aller à la ligne et \t d'ajouter une tabulation
11. • \' permet d'ajouter un ' dans une String par exemple :
12. En POO quasiment tous les types sont des classes. Seuls les types primitifs ne sont pas des classes, par exemples : int, float, double, char, boolean.
13. Pour en savoir plus sur les classes Java existantes allez consulter l'API Java : https://docs.oracle.com/en/java/javase/21/docs/api/
14. On peut définir l'API des classes que vous créez en écrivant la javadoc (cf. https://www.oracle.com/fr/technical-resources/articles/java/javadoc-tool.html).
15. En Java, List et Set sont des interfaces qui représentent des collections, c'est-à-dire des groupes d'objets. Ces interfaces ne définissent pas comment les éléments sont stockés ou gérés, mais elles spécifient les opérations qu'une collection peut effectuer, comme ajouter, supprimer ou accéder à des éléments.
    List est une collection ordonnée où chaque élément a une position spécifique, et les éléments peuvent être dupliqués. Les classes qui implémentent List incluent par exemple ArrayList et LinkedList, chacune ayant ses propres caractéristiques en termes de performance et de structure de données.
    Set, quant à lui, est une collection qui ne permet pas les doublons, c'est-à-dire que chaque élément est unique. HashSet et TreeSet sont des exemples de classes qui implémentent Set, avec des différences dans l'ordre de stockage et la vitesse d'accès.
    Ainsi, en utilisant List ou Set, vous pouvez choisir l'implémentation qui convient le mieux à vos besoins tout en respectant les mêmes règles d'utilisation définies par l'interface.
16. Boucles for-each sur des Collections Java

```
public boolean isEnrolled(String  name, String firstName) {
        for (Student s : students) {
                if (s.getName().equals(name) && s.getFirstName().equals(firstName))
                {
                return true;
                }
        }
return false;
```

17. && exprime un et. L'opérande à droite n'est évalué que si celle de gauche est vraie, donc dans l'exemple ci-dessous que si les noms sont les mêmes.
18. Equals doit être ezdéefini dans les classes pour pouvoir comparer 2 instances de la classe.
19. Overloading (Surcharge de méthodes) :
    Overloading se produit lorsque vous avez plusieurs méthodes dans une même classe qui portent le même nom mais ont des paramètres différents (type, ordre ou nombre de paramètres différents). Les méthodes surchargées ont donc le même nom mais des signatures de méthode différentes. La résolution de la méthode à appeler se fait au moment de la compilation en se basant sur les arguments passés

20. L'héritage (en anglais inheritance) est un principe propre à la programmation orientée objet, permettant de créer une nouvelle classe à partir d'une classe existante. Le nom d'"héritage" (pouvant parfois être appelé dérivation de classe) provient du fait que la classe dérivée (la classe nouvellement créée) contient les attributs et les méthodes de

sa superclasse (la classe dont elle dérive, dont elle hérite). L'intérêt majeur de l'héritage est de pouvoir définir de nouveaux attributs et de nouvelles méthodes pour la classe dérivée, qui viennent s'ajouter à ceux et celles héritées.
Par ce moyen on crée une hiérarchie de classes de plus en plus spécialisées. Cela a comme avantage de ne pas avoir à repartir de zéro lorsque l'on veut spécialiser une classe existante. De cette manière il est possible d'acheter dans le commerce des librairies de classes, qui constituent une base, pouvant être spécialisées (on comprend encore un peu mieux l'intérêt pour l'entreprise qui vend les classes de protéger les données membres grâce à l'encapsulation...).[https://web.maths.unsw.edu.au/~lafaye/CCM/poo/heritage.htm]