Programmation Objet Avancée - TP6

—o000o——o000o—

Collections, exceptions, associations entre objets

1 Gestion des étudiants

À travers cet exercice, nous voulons modéliser la gestion des étudiants d'une promotion. On considère des modules, correspondant à une matière enseignée, et des étudiants inscrits à plusieurs cours selon le diagramme de classes simplifié suivant.

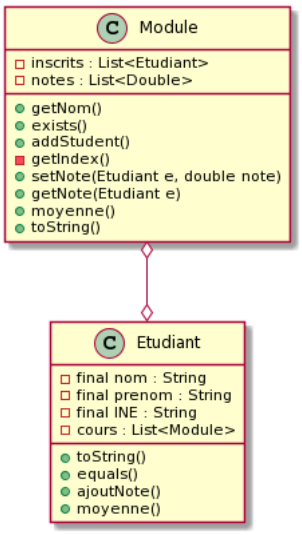

Figure 1: Diagramme de classes simplifié

1.1 Classe `Etudiant`

Nous collectons les données suivantes caractérisant un étudiant : son nom, son prénom, son numéro unique d'étudiant (INE) ainsi que la liste des modules dans lesquel l'étudiant est inscrit.

1. créer une classe `Etudiant` avec les champs demandés avec un constructeur qui initialise la liste des cours suivis à la liste vide ;
2. écrire une méthode `equals(Etudiant e)` qui permet de savoir si deux étudiants sont les mêmes. On comparera pour cela leur nom, leur prénom et leur INE, même si dans la réalité, l'INE est supposé unique et peut servir de clé de comparaison de manière fiable ;
3. écrire une méthode `register(Module m)` qui permet de déclarer le module `m` comme suivi par l'étudiant ;
4. écrire une première version de la méthode `toString()` qui retourne une chaîne de caractères contenant les nom, prénom et INE de l'étudiant. On améliorera cette méthode par la suite, quand la classe `Module` sera implémentée, pour y ajouter la liste des cours suivis.

Solution :

```java
package poo.tp6;

import java.util.ArrayList;
import java.util.List;

public class Etudiant {
    private final String nom, prenom, INE;
    private List<Module> cours;

    public Etudiant(String _nom, String _prenom, String _INE){
        nom = _nom;
        prenom = _prenom;
        INE = _INE;
        cours = new ArrayList<Module>();
    }

    public boolean equals(Etudiant e){
        return nom.equals(e.nom) && prenom.equals(e.prenom) && INE.
           equals(e.INE);
    }

    public String toString(){
        StringBuffer sb = new StringBuffer(nom + ", " + prenom + ", " +
            INE + "\n");
        for (Module m : cours){
            sb.append(m.toString() + "\n");
        }
        return sb.toString();
    }

    public void ajoutNote(String module, double note) throws
       studentNotFoundException {
        boolean trouve = false;
        int i = 0;
        while (!trouve && i < cours.size()){
            if (cours.get(i).getNom().equals(module)){
                trouve = true;
                cours.get(i).addStudent(this); // add student if not
                   registered in the course
                cours.get(i).setNote(this, note); // set her/his grade
            } else {
                i++;
            }
        }
        if (!trouve) System.out.println("Module non trouve"); // change
            for an exception
    }

    public double moyenne() throws studentNotFoundException{
        double sum = 0;
        int cpt = 0;
        for (Module m : cours){
```

```
48              double note = m.getNote(this);
49              if (note >= 0) {
50                  sum += note;
51                  cpt ++;
52              }
53
54          }
55          if (cpt == 0) throw new ArithmeticException("Pas de note");
56          else return sum / cpt;
57      }
58
59      public void register(Module m){
60          cours.add(m);
61      }
62  }
```

java/Etudiant.java

1.2 Classe `Module`

Un module représente un cours et possède un intitulé (son nom de module), un nom de professeur, une liste d'étudiants inscrits et une liste de notes qui contient pour chaque étudiant sa note (unique dans notre exemple) à ce cours. Pour des raisons de simplicité on considérera par la suite que l'ordre des notes correspond à l'ordre des étudiants. Vos méthodes devront garantir que cet ordre est toujours respecté. Si vous avez le temps, nous vous encourageons à regarder comment il serait possible de gérer ces notes plus efficacement avec une structure de type `Map` plutôt que 2 structures de type `List`.

1. créer la classe Module avec ses différents champs ainsi qu'un constructeur permettant de définir le nom du module et du professeur. Ces informations n'étant pas susceptibles de changer, vous êtes encouragés à les déclarer avec le mot-clé `final`. Les deux listes (d'étudiants et leurs notes) sont vides initialement ;

2. écrire une méthode `getNom()` qui retourne le nom (ou l'intitulé) du module ;

3. écrire une méthode `exists(Etudiant e)` qui retourne vrai si et seulement si l'étudiant `e` apparaît dans la liste des inscrits pour ce module ;

4. écrire la méthode **privée** `getIndex(Etudiant e)` qui renvoie, si il existe dans la liste des inscrits, l'indice auquel apparaît l'étudiant `e` ;

5. ajouter une méthode `addStudent(Etudiant e)` qui permet d'ajouter, s'il n'existe pas déjà, un étudiant à la liste des inscrits à ce cours ; attention, de faço,n à garantir que la liste de notes fait la même taille que la liste d'étudiants, il fau également lui ajouter une valeur, par exemple -1 par défaut avant qu'une note ne soit affectée. Lors du calcul de la moyenne pour ce module il faudra bien prendre garde à ne pas considérer les notes inférieures à 0. En parallèle, la méthode doit aussi ajouter ce cours à l'étudiant à l'aide de sa méthode `register` dédiée ;

6. écrire une première version de la méthode `toString()` pour un module, permettant de retourner son intitulé, le nom du professeur et la liste des étudiants inscrits ; nous reviendrons sur cette méthode plus tard pour ajouter la moyenne des notes des étudiants.

Solution :

```
1  package poo.tp6;
2
```

```java
import java.util.ArrayList;
import java.util.List;

public class Module {
    private final String nom, professeur;
    List<Etudiant> inscrits;
    List<Double> notes;

    public Module(String _nom, String _prof){
        nom = _nom;
        professeur = _prof;
        inscrits = new ArrayList<Etudiant>();
        notes = new ArrayList<Double>();
    }

    public String getNom(){
        return nom;
    }

    public void addStudent(Etudiant e){
        if (!exists(e)) {
            this.inscrits.add(e);
            this.notes.add(-1.0); // to ensure that inscrits and notes
                have the same size
            e.register(this);
        }
    }

    /**
     * Return true iif student e has a grade for that particular class
     * @param e: student
     * @return true iif student e has a grade for that particular class
         and false otherwise
     */
    public boolean exists(Etudiant e){
        return inscrits.contains(e); // need e to define equals!
    }

    /**
     * Returns the index of student e iif he/she exists in the list
     * @param e : student
     * @return
     */
    private int getIndex(Etudiant e){
        return inscrits.indexOf(e); // need e to define equals!
    }

    /**
     * Set a grade for a student e
     * @param e
     * @param note
     * @throws studentNotFoundException
     */
```

```java
    public void setNote(Etudiant e, double note) throws
        studentNotFoundException{
        if (exists(e)) notes.set(getIndex(e), note);
        else throw new studentNotFoundException("Etudiant inconnu");
    }

    /**
     * Retrieve a grade for a student e
     * @param e
     * @return
     * @throws studentNotFoundException
     */
    public double getNote(Etudiant e) throws studentNotFoundException{
        if (exists(e)) return notes.get(getIndex(e));
        else throw new studentNotFoundException("Etudiant inconnu");
    }

    /**
     * return the average grade for this students' class
     * @return
     */
    public double moyenne() throws ArithmeticException {
        double sum = 0;
        int cpt = 0;
        for (double val : notes) {
            if (val >= 0) {     // because by default grades = -1 when
                adding a new student
                sum += val;
                cpt++;
            }
        }
        if (cpt == 0) throw new ArithmeticException("Pas de note");
        else return sum / cpt;
    }

    /**
     * Returns a String representation of a class
     * @return
     */
    public String toString(){
        StringBuffer sb = new StringBuffer("Module " + nom + ", sous la
            responsabilite du Pr. " + professeur);
        sb.append("\n");
        sb.append(inscrits.size() + " etudiants\n avec une moyenne de "
            + moyenne());
        sb.append("\n");
        return sb.toString();
    }
}
```

java/Module.java

1.3 Gestion des erreurs

Lors de la gestion des notes d'un étudiant dans un module, il est possible que l'on demande la modification ou la récupération d'une note pour un étudiant qui n'existe pas dans la liste des inscrits. Dans ce cas, nous voulons gérer proprement cette erreur en définissant notre propre `Exception`.

1. écrire la classe `studentNotFoundException` qui hérite de `Exception` et qui permet de gérer de tels cas. On permettra notamment de définir un message d'erreur personnalisé lors de la récupération d'une erreur.

Solution :

```java
package poo.tp6;

public class studentNotFoundException extends Exception {
    public studentNotFoundException() {
        super();
    }

    public studentNotFoundException(String s) {
        super(s);
    }
}
```

java/studentNotFoundException.java

1.4 Retour sur la classe `Module`

On peut désormais finaliser notre classe `Module`.

1. écrire une méthode `setNote(Etudiant e, double note)` qui permet de modifier la note de l'étudiant `e` s'il est inscrit au module. Dans le cas contraire, une exception `studentNotFoundException` doit être retournée ;
2. écrire une méthode `getNote(Etudiant e)` qui permet de récupérer la note de l'étudiant `e` s'il est inscrit au module. Dans le cas contraire, une exception `studentNotFoundException` doit être retournée ;
3. proposer une méthode `moyenne()` qui retourne la moyenne des notes pour les étudiants de ce module.
4. compléter la méthode `toString()` pour afficher en plus la moyenne des inscrits pour ce cours.

1.5 Retour sur la classe `Etudiant`

Il est désormais possible de finaliser à son tour la classe représentant un étudiant. Pour cela :

1. écrire une méthode `ajoutNote(String module, double note)` qui, connaissant le nom d'un module (son intitulé), le cherche dans la liste des cours suivis par l'étudiant, et s'il est trouvé, éventuellement ajoute l'étudiant à ce cours et enregistre sa nouvelle note. Dans le cas où aucun module ne correspond, on se contentera d'afficher un message d'erreur approprié, mais si vous avez un peu de temps, nous vous encourageons à mettre en place une nouvelle exception dédiée ;
2. compléter la méthode `toString()` de telle sorte qu'elle affiche la liste des modules auxquels l'étudiant est inscrit.

1.6 Pour finir

Il vous reste désormais à produire une classe de test pour vos méthodes nommée `GestionEtudiants` :

1. définir une liste nommé `groupe` qui contient au moins 5 étudiants ;
2. définir une liste nommée `cours` qui contient au moins 2 modules. Réaliser une répartition des étudiants par cours (par exemple 3 étudiants parmi les 5 dans chaque cours) ;
3. pour chaque cours, parcourir tous les étudiants inscrits et leur ajouter une note générée aléatoirement entre 0 et 20 inclus ;
4. parcourir les cours et les afficher à l'aide de la méthode `toString()` ;
5. parcourir les étudiants et les afficher à l'aide de la méthode `toString()` ;

Solution :

```java
package poo.tp6;

import java.util.ArrayList;
import java.util.List;

public class GestionEtudiant {
    public static void main(String[] args) throws
        studentNotFoundException {
        // generating students
        List<Etudiant> groupe = new ArrayList<Etudiant>();
        groupe.add(new Etudiant("Song", "Al", "ABC123"));
        groupe.add(new Etudiant("SmallShirt", "Pat", "XYZ321"));
        groupe.add(new Etudiant("Spike", "Nic", "NYXL1046"));
        groupe.add(new Etudiant("Lee", "Bruce", "KRT999"));
        groupe.add(new Etudiant("Jackson", "Peter", "HOBT123"));

        // generating courses
        List<Module> cours = new ArrayList<Module>();
        cours.add(new Module("Algo", "John Doe"));
        cours.add(new Module("BD", "Alice Bob"));

        // registering students to the courses
        cours.get(0).addStudent(groupe.get(0));
        cours.get(0).addStudent(groupe.get(1));
        cours.get(0).addStudent(groupe.get(2));

        cours.get(1).addStudent(groupe.get(2));
        cours.get(1).addStudent(groupe.get(3));
        cours.get(1).addStudent(groupe.get(4));

        // generating grades
        for (Module m : cours){
            for (Etudiant e : m.inscrits){
                e.ajoutNote(m.getNom(), (int) (Math.random() * 21));
            }
        }

```

```
37          // printing informations
38          for (Module m : cours) System.out.println(m);
39
40          for (Etudiant e : groupe) System.out.println(e);
41      }
42  }
```

java/GestionEtudiant.java