Programmation Objet Avancée - TP9

—o000o——o000o—

Révisions

1 Gestion de véhicules (2018)

On s'intéresse à la gestion d'un parc de véhicules composés de **voitures** et de **motos**. Tous les véhicules ont pour attribut commun une *marque*, un *modèle*, un *numéro d'immatriculation*, et un *identifiant unique* de compagnie d'assurance. Les véhicules, quels que soient leur type, ne sont pas assurés à leur début et l'identifiant est donc vide. Utilisez des attributs protected pour vous affranchir de la définition des accesseurs (getters) et des mutateurs (setters) dans vos définitions.

1. Créer une classe `Vehicule` pour représenter tous ces types de véhicules. Quelle type de classe allez-vous utiliser ?
2. Est-il possible de définir un constructeur pour votre véhicule ? Quel serait son intérêt dans notre cas d'usage ?
3. Ajouter à cette classe `Vehicule`, une méthode d'affichage `String toString()` qui affiche l'ensemble des champs de l'objet. Si le véhicule n'est pas assuré, l'affichage doit indiquer "non assuré".
4. Les **voitures** possèdent en plus un attribut décrivant le nombre de places (2, 4 ou 7) et un attribut booléen qui indique si le véhicule roule avec une énergie verte (hybride, électrique ...). Créer une classe `Voiture` en réutilisant autant que possible les propriétés et les méthodes de `Vehicule`. Vous préciserez son constructeur et sa méthode d'affichage qui doit également indiquer les valeurs des attributs propres aux voitures.
5. Une **moto** est un véhicule pour lequel on stocke une information de cylindrée. Comme pour les voitures, créer une classe `Moto` avec une cylindrée et implémenter une méthode d'affichage dédiée.
6. On souhaite pouvoir ajouter la fonctionnalité transversale aux classes `Voiture` et `Moto` pour les assurer. Quel mécanisme Java pouvez-vous utiliser pour cela ? Déterminer par ce biais un contrat qui force les classes `Voiture` et `Moto` à déclarer une méthode `void assurer (String assureur)` qui permet de définir l'identifiant de compagnie d'assurance d'un véhicule. A quel niveau utiliser cette interface pour que les voitures et motos puissent indifféremment être assurées ?
7. Créer une classe `Gestion` qui contient une structure de liste nommée `parc` pouvant contenir un nombre indéterminé de véhicules de tous types. Vous préciserez pour cela le type générique le plus adapté à la gestion de tous types de véhicules.
8. Créer un constructeur pour la classe `Gestion` qui ajoute 2 voitures et 2 motos à la liste des véhicules.
9. À partir de cette structure de liste, créer une méthode $\texttt{non}_assures()quiretournelalistedesvhiculesprsentsdansleparc$
10. Proposer une méthode `assurance()` qui prend en argument une liste de véhicules et un identifiant d'assureur et déclare cet assureur pour l'ensemble des véhicules de la liste.

2 Types génériques (2018)

Indiquez pour chaque instruction si elle génère une erreur ou bien si elle est acceptée par Java.

1. `class A<T> extends B<T extends Number> {...}`
2. `class A extends B<T> {...}`

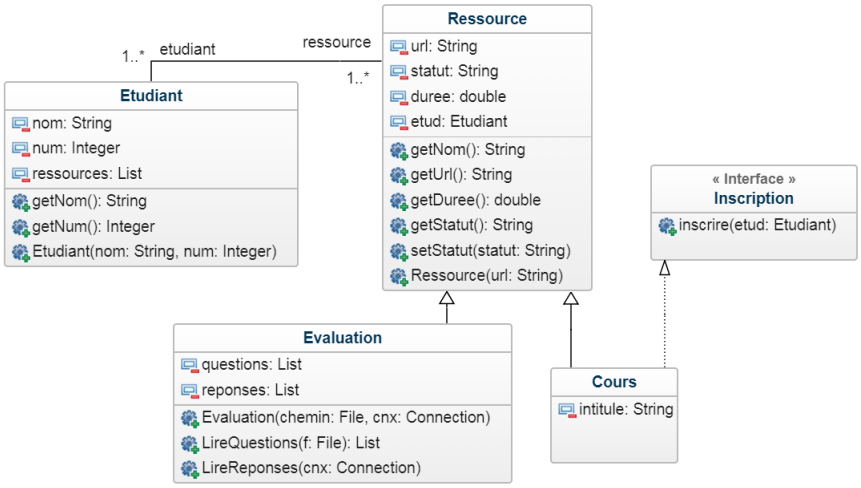

Figure 1: Diagramme UML des classes

3. `class A<T,U> extends B<T> {...}`
4. `class A<T extends Number> extends B<T> {...}`

3 Site d'apprentissage en ligne (2018)

Nous souhaitons développer une application pour gérer un site d'apprentissage en ligne. Le site propose des `Ressources` en ligne sous la forme de `Cours` ou `Evaluation`. Un `Etudiant` peut s'inscrire à certaines `Ressources`. Dans ce cas, la ressource apparaît dans la liste des ressources de l'étudiant mais, de manière symétrique, l'étudiant apparaît dans la liste des membres inscrits à la ressource.

Important pour chacune des questions ci-après vous pouvez ajouter les méthodes ou champs qui vous paraissent nécessaires même si elles ne sont pas demandées dans l'énoncé. De même, n'implémentez pas nécessairement toutes les méthodes du diagramme UML si vous n'en avez pas l'utilité. Veillez à bien respecter le principe d'encapsulation des données.

1. Écrire la classe abstraite `Ressource` qui est décrite par une `URL` (`String`), une durée exprimée en nombre d'heures (`double`) et un statut (`String`). La classe définit également deux constantes qui définissent les états possibles du statut : `CONSULTEE` et `NON_CONSULTEE`. La classe possède un constructeur qui prend uniquement une `URL` en argument et fixe la durée par défaut à 1 heure et le statut à la valeur `NON_CONSULTEE`.

2. Écrire la classe `Etudiant` qui représente un client du site par son `nom` (`String`), un numéro d'étudiant `num` (`int`) et la liste des `Ressource`(s) auxquelles il/elle est inscrit. Proposer un constructeur pour cette classe qui prend en argument un nom et un numéro d'étudiant. La liste des ressources est initialement vide.

3. Écrire l'interface Inscription qui déclare une fonction `void inscrire(Etudiant)`.

4. Écrire le début de la classe `Evaluation` qui hérite de `Ressource` et contient une liste de questions ainsi qu'une liste de réponses attendues. Les questions et les réponses sont des chaînes de caractères (`String`). Ajouter un constructeur qui prend en argument un chemin de type `File` vers un fichier texte ainsi qu'un objet de type `Connection` pour faire un lien avec une base de données relationnelle.

5. Dans la classe `Evaluation` ajouter une méthode `List<String> lireQuestions(File)` qui initialise la liste de questions à partir du contenu du fichier texte dont la référence a été donnée dans le constructeur. Ce fichier contient sur chaque ligne : un identifiant de question, un caractère

espace et le texte de la question. Votre méthode doit insérer le texte des questions dans la liste de questions de `Evaluation` et retourner la liste des identifiants de questions au format `List<String>`.

6. Construire une table dans votre gestionnaire de base de données préféré (`MySQL`) contenant deux champs : un identifiant de question `idQ (VARCHAR(10))` et un texte de réponse attendue à la question `repQ (VARCHAR(50))`. Préparer le code nécessaire pour vous connecter à cette table de base de données et obtenir un objet `Connection` valide.

7. Dans la classe `Evaluation` ajouter une méthode `lireReponse(Connection, List<String>)` qui initialise la liste des réponses correctes à partir d'une liste des identifiants de questions et d'une connexion JDBC à une base de données relationnelle contenant les réponses.

8. Écrire la classe `Cours` qui hérite de `Ressource` et implémente l'interface `Inscription`. Proposer un constructeur ainsi que la méthode `inscrire` qui permet d'ajouter un étudiant à ce cours mais également de déclarer ce cours dans la liste des ressources de l'étudiant. Lors de l'inscription du premier étudiant le statut de la ressource doit passer de `NON_CONSULTEE` à `CONSULTEE`. Ajouter une méthode d'affichage qui liste les noms des étudiants inscrits à ce cours.

9. Proposer une classe de test qui construit plusieurs ressources de type cours et une évaluation donc les questions et les réponses seront lues respectivement dans un fichier texte csv et une base de données.