Examen de Programmation Java – ING1 EISTI 2009-2010

L'architecture (nom des répertoires) de votre projet Java est imposée de la manière suivante :

- sources
- build
    - bytecode
    - jar
- build.xml

Déposer votre projet (uniquement sources Java et fichier ant) dans le répertoire **rendu-java**.

La javadoc de l'API 1.6 est disponible dans le répertoire **/usr/share/doc/sun-java6-jdk/html**

Pour accéder à la version la plus récente d'Eclipse, tapez eclipse-3.6

Rappel du barème des pénalités (à ramener au prorata de chaque exercice) :

- code non compilable : -10 (moitié de la note)
- warning (annotation @deprecated interdite) : -2 (un 10ème)
- norme de programmation Java non respectée : -5 (un quart)
- pas de commentaires : -5 (un quart) (**)
- non respect des consignes (noms de classe/package, fichiers déposés) : -2

NB : (**) le temps imparti étant relativement court, votre capacité à commenter votre code sera évaluée uniquement sur la classe Pool (exercice 2). Vous êtes dispensé (exceptionnellement) des commentaires habituels pour les autres classes.

PJ : vous trouverez à la racine de votre compte exam-manager :

- le sujet en PDF
- le fichier source de la classe Student à placer dans votre projet (toute modification du fichier est interdite) ;
- un fichier de données studs.obj pour tester votre application

Introduction

Vous devez réaliser une application qui construit un groupe d'étudiants de l'EISTI. Une application déjà réalisée a permis de collecter les futurs étudiants du groupe sous la forme d'objets Student (du paquetage org.example.teaching.data) et de les sauver sous une forme sérialisée dans un fichier objet.

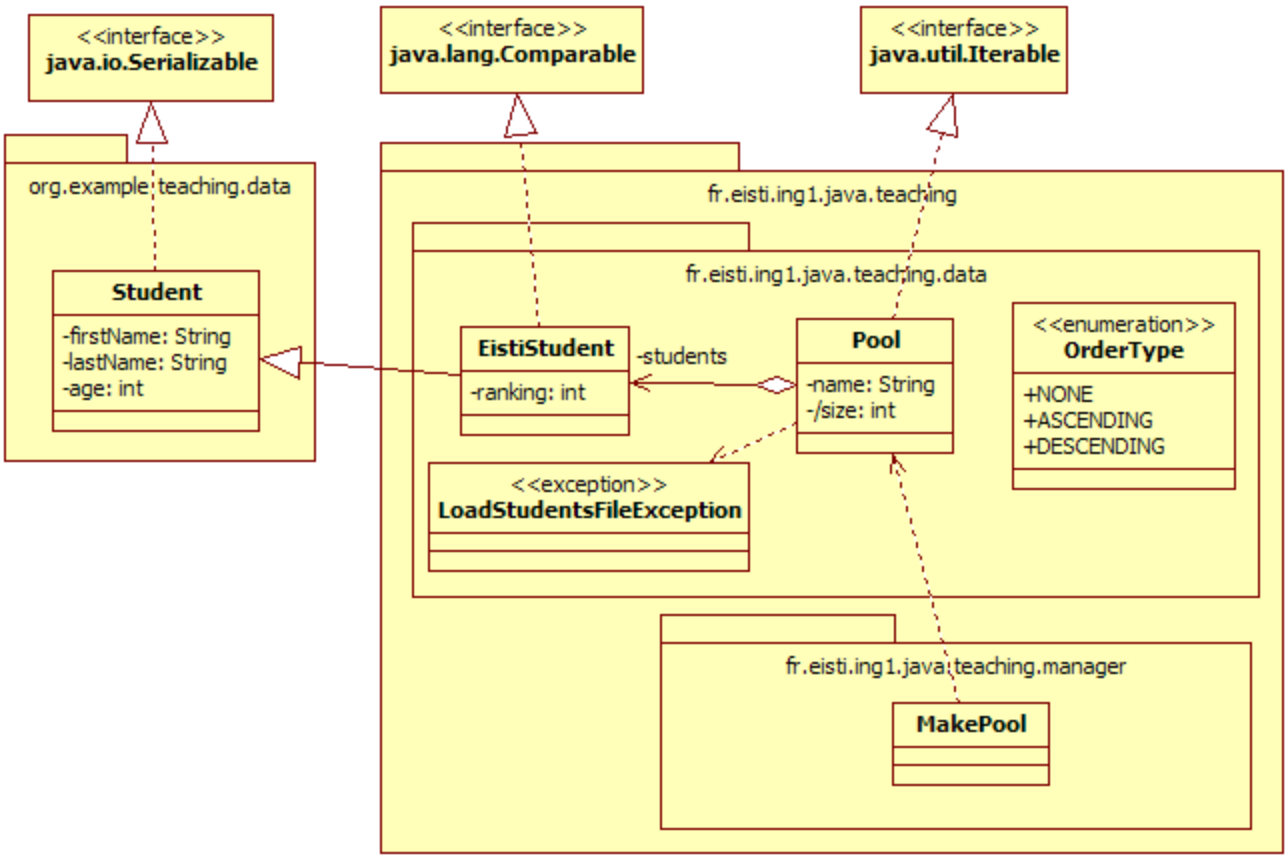
Pour pouvoir utiliser ces étudiants dans le système d'information de l'EISTI, il faut leur ajouter un attribut ranking, éliminer les éventuels doublons dus à des saisies multiples et fabriquer un groupe d'étudiants ainsi complétés (Pool).

Pour pouvoir charger le fichier de données initiales, il n'est pas possible de modifier la classe Student pour ajouter l'attribut ranking. Il est décidé de développer la classe EistiStudent pour compléter la classe Student.

Un groupe d'étudiant ainsi construit pourra être parcouru de 3 manières :

- NONE : sans précision d'ordre, tel que le groupe a été construit ;
- ASCENDING : parcours des étudiants triés suivant l'ordre (ranking, nom, prénom, age) ascendant ;
- DESCENDING : parcours inverse du précédent.

Le diagramme UML suivant vous permet d'avoir un aperçu global du système :



Exercice 1 – Classe EistiStudent

Ecrire la classe EistiStudent (du paquetage fr.eisti.ing1.java.teaching.data) qui spécialise la classe Student (du paquetage org.example.teaching.data) :

- en ajoutant un attribut ranking ;
- avec un constructeur complet prenant en paramètre ses 4 propriétés ;
- avec un constructeur prenant en paramètre un objet Student pour construire ses 3 premiers attributs et initialisant le ranking à 0 ;
- avec 4 accesseurs en lecture et uniquement 1 en écriture pour le ranking ;
- avec une méthode equals retournant vrai si les 4 propriétés sont égales
- avec une méthode toString permettant de représenter un étudiant sur le modèle suivant :

  Nom Prénom (âge) #ranking
  Dupont Emile (21) #12
- implémentant l'interface java.lang.Comparable pour définir l'ordre entre 2 objets EistiStudent avec dans l'ordre :
  - comparaison du ranking en priorité (ordre ascendant numérique) ;
  - comparaison du nom de famille (lastName) par ordre alphabétique ;
  - comparaison du prénom (firstName) par ordre alphabétique ;
  - comparaison de l'âge (ordre ascendant numérique).

Exercice 2 – Classe Pool

Ecrire la classe Pool (du paquetage fr.eisti.ing1.java.teaching.data) permettant de définir un groupe d'étudiant de type EistiStudent (du même paquetage) :

- avec un nom de groupe (name) ;
- avec un constructeur prenant en paramètre le nom du groupe et initialisant un ensemble vide d'étudiants ;

- avec un constructeur prenant en paramètre le nom du groupe et un nom de fichier contenant des objets Java sérialisés de type Student (du paquetage org.example.teaching.data) ; le constructeur doit lire le fichier et construire pour chaque objet lu un étudiant EistiStudent à ajouter dans le groupe. L'ajout devra éliminer les éventuels doublons contenus dans le fichier ; tout problème de chargement propagera l'exception LoadStudentsFileException (et uniquement celle-là), encapsulant l'exception source du problème ;
- avec un accesseur en lecture pour le nom du groupe et sa taille ;
- avec une méthode d'ajout (sans doublon) et de suppression d'un étudiant de type EistiStudent ;
- avec une méthode toString permettant de connaître le nom et la taille du groupe ;
- implémentant l'interface java.util.Iterable permettant de parcourir les étudiant du groupe (ordre quelconque) ;

et au choix :

- pour disposer de plusieurs parcours possibles, on surcharge l'itérateur de l'interface Iterable en ajoutant un paramètre de type OrderType (à définir dans le paquetage fr.eisti.ing1.java.teaching.data) :

  `Iterator<EistiStudent> iterator(OrderType order)`

  Le paramètre *order* précise l'ordre du parcours (NONE, ASCENDING, DESCENDING) ; l'itérateur par défaut (celui de l'interface Iterable) assurera le parcours NONE ;
- ajouter une méthode de signature `List<EistiStudent> trierEtudiant(OrderType)` qui permet d'obtenir la liste des étudiants du groupe triés suivant l'ordre passé en paramètre (cf introduction).

NB : penser à utiliser les outils à votre disposition du framework des collections (tri, inversion de liste, etc…)

Exercice 3 – Application MakePool

Ecrire l'application Java MakePool (du paquetage fr.eisti.ing1.java.teaching.manager) qui :

- prend 3 paramètres en ligne de commande :
  1. le nom du groupe à fabriquer ;
  2. le nom du fichier contenant les objets de type Student (du paquetage org.example.teaching.data) ; un exemple d'un tel fichier vous ait fourni en PJ (studs.obj) ;
  3. le nom du fichier texte qui contiendra le groupe d'étudiants fabriqué et trié par l'application ;
- construit un groupe d'étudiants de l'EISTI (Pool) à partir des 2 premiers paramètres ;
- affecte un ranking aléatoire (entre 0 et 20) à chaque étudiant du groupe ; vous pouvez utiliser à cet effet la méthode java.lang.Math.random ou la classe java.util.Random ;
- affiche sur la console le groupe avec l'ordre de parcours ASCENDING : nom et taille du groupe puis les étudiants triés (un par ligne) ;
- sauve dans un fichier texte (3ème paramètre) le groupe (même contenu que l'affichage console) avec l'ordre de parcours DESCENDING ;
- tout problème éventuel sera capté et signalé sur la console à l'utilisateur avant de quitter le programme.

Exercice 4 – Gestion du projet Java et exécutable

Ecrire un fichier **build.xml** qui permet de :

- compiler l'ensemble de vos sources compilables (si vous avez une classe incomplète, l'exclure des fichiers à compiler) ;
- fabriquer un jar exécutable pour l'application MakePool : **makeStudentPool.jar**
- effacer le bytecode
- effacer tout ce qui est construit

NB : Votre fichier de configuration devra être en adéquation avec l'architecture de fichiers rendue. Le rendu d'un fichier déjà prêt et non adapté au projet du jour ne sera pas noté.