Projet JAVA IR1 2012

Projet EvaluationsEleves/Professeurs

Ce projet Java comporte les objectifs suivants:

1) Diagramme de classes
2) Développer de l'application en mode console avec un jeu de tests fourni
3) Lecture des notes/Eleves/Professeurs dans un fichier Excel sauvegardé au format csv
4) De nouveau affichage en mode console
5) Statistiques diverses via JfreeChart
6) Javadoc des classes développées
7) En option: Interface graphique en Swing

Cahier des charges (CDC):

On cherche à gérer une liste d'élèves avec leurs notes et les professeurs qui les évaluent.

- Chaque **professeur** et chaque **élève** ont comme propriété commune le prénom et le nom.
- Chaque **élève** a les propriétés suivantes:
  - un numéro d'identifiant dont l'unicité n'est pas gérée (prochaine version...)
  - son nom, son prénom
  - un tableau de 10 **évaluations**, sa moyenne
- Pour chaque **évaluation**, sont indiqués la valeur de la note, l'**élève** corrigé et le **professeur** correcteur
- Chaque **élève** ne peut consulter que l'ensemble de ses propres notes ainsi que sa moyenne.
- L'ensemble des élèves est une **promotion** et chaque **élève** connaît sa **promotion**.
- Les élèves d'une **promotion** peuvent être triés par moyenne des notes.
- Chaque **professeur** peut effectuer les opérations suivantes:
  - rechercher un élève dans la **promotion** à partir de son numéro d'identifiant
  - modifier des notes d'un élève recherché

Parmi les classes à développer figurent entre autres les classes:

Eleve, Professeur, Evaluation, Promotion.

Ces classes sont rangées dans un **package notesElevesProfesseurs et d'autres packages si besoin est.**

Question 1:

CDC: *Chaque professeur et chaque élève ont comme* ***propriétés communes*** *le prénom et le nom.*

Quel concept de la programmation objet est approprié à ce point du CDC?

Ecrire le code nécessaire correspondant à ce concept?

Question 2: classes Professeur et Eleve

CDC: *Chaque professeur et chaque élève ont comme attributs communs le prénom et le nom.*

Définir les classes **Professeur** et **Eleve** en tenant compte de vos réponses à la question 1.

Définir les accesseurs en lecture pour le nom et le prénom

Rédéfinir la méthode **toString** telle qu'elle retourne par exemple:

(Soleil, Tournesol) pour le professeur de prénom Soleil et de nom Tournesol.

(Jean, Duval) pour l'élève de prénom Jean et de nom Duval

Question 3: classe Evaluation

CDC: *Pour chaque évaluation, sont indiqués la valeur flottante de la note, l'****élève*** *corrigé et le* ***professeur*** *correcteur.*

Coder la classe **Evaluation** définissant:

- les 3attributs ci-dessus
- un constructeur à 3 paramètres

La classe **Evaluation** redéfinit la méthode **toString** telle qu'elle retourne par exemple:

((Soleil, Tournesol),(Jean, Duval) 12.0)

Ecrire le code de cette méthode.

Question 4: classe Elève (suite)

CDC: *Chaque* ***élève*** *a les attributs suivants:*

- *un numéro d'identifiant*
- *son nom, son prénom*

- *un tableau de 10 évaluations, sa moyenne*

indication: l'identifiant est une constante entière.

Coder la classe **Eleve** définissant:

- une constante de classe **NB_EVALUATIONS** de type entier et valant 10
- un constructeur public à 3 paramètres permettant de donner un nom, un prénom et un identifiant à un élève.
- un accesseur en lecture pour l'identifiant
- une méthode **moyenne** calculant et retournant la moyenne des notes de l'élève référencé par l'instance courante. Si l'élève n'a aucune note, l'**exception IllegalStateException** est lancée.
- la méthode **getCorrecteurs** permettant de ranger dans une instance de la classe **HashSet**, l'ensemble des correcteurs ayant évalué un élève. Elle retourne cette collection.

  La signature est : **public Set<Prof> getCorrecteurs();**

rappel: **HashSet** est une classe implémentant l'interface **Se**t, disponible dans le package **java.util**.

Méthode d'ajout de l'interface Set, où E désigne un type générique:

boolean add(E e)

Adds the specified element to this set if it is not already present (optional operation). More formally, adds the specified element `e` *to this set if the set contains no element* `e2` *such that* `(e==null?e2==null:e.equals(e2))`. *If this set already contains the element, the call leaves the set unchanged and returns* `false`. *In combination with the restriction on constructors, this ensures that* ***sets never contain duplicate elements.***

La classe **HashSet** rédéfinit une méthode **toString** de telle sorte qu'une collection de professeurs sera affichée de la manière suivante:

[(Max, La Menace), (Soleil, Tournesol)]

La classe **Eleve** redéfinit la méthode **toString** telle qu'elle retourne par exemple:

(Jean, Duval) id: 1 notes: 12.0 6.0 15.0 moyenne = 11.0

correcteur(s): [(Max, La Menace), (Soleil, Tournesol)]

Ecrire le code de cette méthode.

Question 5: classe Promotion

***CDC**: L'ensemble des élèves est une **promotion**.*

Coder la classe **Promotion** définissant entre autres:

- un attribut **nom** de type String
- un attribut **eleves de type List** générique. Une promotion contient ainsi une collection d'élèves.
- Un constructeur à un paramètre
- des accesseurs en lecture et en éciture pour le **nom** de la promotion
- un accesseur en lecture **getEleves**. Justifier sa signature en commentaires.
- une méthode **rechercher** qui recherche un élève dans la collection des élèves selon son identifiant passé en paramètre. Si l'élève recherché est trouvé, cette méthode retourne l'élève trouvé, sinon retourne un référence nulle.

Question 6: classe Eleve (suite)

***CDC:** L'ensemble des élèves est une **promotion** et chaque **élève** connaît sa **promotion**.*

Compléter les classes **Promotion** et **Eleve** en cons équence.

Notamment la méthode **toString** de la classe **Eleve** doit afficher le nom de la promotion d'un élève.

Question 7: classe Professeur (suite)

***CDC:** Chaque professeur peut effectuer les opérations suivantes:*

- *rechercher un élève dans une promotion à partir de son numéro d'identifiant*

Ecrire une méthode **rechercher** qui recherche un élève dans une promotion donnée selon son identifiant passé en paramètre. Si l'élève recherché est trouvé, cette méthode retourne l'élève trouvé, sinon retourne un référence nulle.

Question 8: classe Professeur (suite)

***CDC:** Chaque **professeur** peut effectuer les opérations suivantes: .*

- *modifier des notes d'un élève recherché*

Ecrire une méthode **setNote** qui modifie la ième note du tableau de notes d'un élève:
La promotion, l'identifiant de l'élève, la valeur de la note et l'indice i sont passés en paramètres de cette méthode.
A partir du numéro d'identifiant de l'élève, cette méthode recherche cet élève et modifie si possible une note.
Si l'élève recherché n'existe pas, l'**exception IllegalStateException** est lancée.
Si l'élève existe, la note d'indice i du tableau notes de l'élève corrigé est modifiée: si cette note d'indice i est vide alors elle est créé sinon sa valeur est modifiée avec la valeur passée en paramètre. Le professeur correcteur est le professeur référencé par l'instance courante.

Question 9 : classes Eleve et Promotion (suite)

CDC: Les élèves d'une promotion peuvent être triés par moyenne des notes.

- Les élèves sont comparer par ordre croissant puis décroissant selon leur moyenne.
  Comment faut-il compléter la classe **Eleve**? Ecrire le code nécessaire.
- Dans la classe **Promotion**, coder les méthodes qui trient les élèves de la promotion par ordre croissant puis décroissant de moyenne.

Question 10 : classe Main

Ecrire une classe Main contenant une fonction main de manière à:

- Créer 4 élèves et 2 professeurs
- Ranger les élèves dans leur promotion
- Mettre des notes aux élèves
- Afficher tous les élèves d'une promotion
- Afficher un élève avec son nom, sa promotion, ses notes et ses correcteurs.
- Rechercher un élève par son identifiant et l'afficher
- Afficher un élève avec son nom et sa moyenne
- Trier les élèves par ordre croissant puis décroissant de leur moyenne
- Afficher tous les élèves d'une promotion après les tris