# Projet EvaluationsEleves/Professeurs

## I VERSION 1 : Mode console avec un jeu de tests

**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 est à gérer
  - son nom, son prénom
  - sa date de naissance (jour mois année), une classe Date est à développer
  - ses **évaluations,** leur moyenne et leur médiane
- Pour chaque **évaluation**, sont indiqués la matière concernée, 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 leur moyenne et leur médiane.
- L'ensemble des élèves est une **promotion** et chaque **élève** connaît sa **promotion**.
- Les élèves d'une **promotion** peuvent être classés par moyenne des notes, par médiane des notes. Des classements par matière sont également possibles.
- 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?

*(Réponse en commentaires dans le code)*

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 matière concernée, la valeur flottante de la note, l'**élève** corrigé et le **professeur** correcteur.*

Coder la classe **Evaluation** définissant:

- les 4 attributs ci-dessus
- un constructeur à 4 paramètres

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

*((Jean, Duval) (Soleil, Tournesol) mathématiques 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*
- *ses évaluations, leur moyenne et leur moyenne*

*indication*: l'identifiant est une constante entière et tous les élèves ont des identifiants uniques (le recours à des variables de classes, mot-clé static, peut être envisagé).

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 d'un élève.
  Si l'élève n'a aucune note, l'**exception IllegalStateException** est lancée.
- une méthode **mediane** calculant et retournant la médiane des notes d'un élève.
  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: mathématiques 12.0 français 18 mathématiques 6.0 anglais 15.0*

*moyenne = 12.75*

*mediane = 13.5*

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

Ecrire le code de cette méthode.

Indication : la classe **Eleve** doit implémenter les méthodes **equals** et **hashcode** pour les besoins de certaines méthodes de la classe HashSet entre autre.

Il est en est de même des autres classes.

Attention toutefois à la circularité des appels à ces méthodes. Vous devrez modifier le code généré afin de court-circuité la circularité éventuellement observée.

## 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** d'un container générique. Une promotion contient ainsi une collection d'élèves.
- Un constructeur à un paramètre
- des accesseurs en lecture et en écriture pour le **nom** de la promotion
- un accesseur en lecture **getEleves**. Justifier sa signature en commentaires dans le code. Attention à ne pas violer le principe d'encapsulation.
- 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 conséquence.

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 des évaluations d'un élève*

Ecrire une méthode **setNote** qui modifie la ième note 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 et si la note d'indice i existe aussi alors est modifiée.
_ si l'élève existe et si la note d'indice i n'existe pas alors elle est crée et l'évaluation est ajoutée aux évaluations existantes.

## Question 9 : classes Eleve et Promotion (suite)

***CDC :*** Les élèves d'une promotion peuvent être classés selon la moyenne et la médiane de leurs notes.

- Les élèves sont classés par ordre croissant puis décroissant selon la moyenne et la médiane de leurs notes. Comment faut-il compléter la classe **Eleve**? Ecrire le code nécessaire.
- Dans la classe **Promotion**, coder les méthodes qui classent les élèves de la promotion par ordre croissant puis décroissant de moyenne et de médiane.

## Question 10 : classes de tests

Ecrire plusieurs classes de tests, dans un package ***test***, contenant une méthode **main** de manière à :

- Créer plusieurs élèves et plusieurs professeurs
  *Les données sont codées en dur dans des classes de tests.*
- Ranger les élèves dans leur promotion
- Mettre des notes aux élèves, indiquer les correcteurs
- Afficher un élève avec son nom, sa promotion, ses correcteurs, ses notes, leur moyenne et leur médiane
- Afficher tous les élèves d'une promotion
- Rechercher un élève par son identifiant puis l'afficher (pas d'affichage dans la recherche!)
- Classer les élèves par ordre croissant puis décroissant de leur moyenne et de leur médiane
- Afficher tous les élèves d'une promotion selon les classements effectués.

## II VERSION 2 : Mode console avec un jeu de tests

Les données sont lues dans des fichiers au format csv.

L'utilisation de la classe ***Scanner*** est requise. Consulter la Javadoc.

La définition de la grille de lecture du fichier est laissée à votre jugement.

Des données pouvant être modifiées par l'application (notamment des notes), leur sauvegarde, au même format csv, est requis.

## III VERSION 3 : Mode console  statistiques diverses avec un jeu de tests

Les données sont codées en dur dans des classes de tests ou bien lues dans des fichiers au format csv.

Le bulletin de notes d'un élève est à modéliser.

Less évaluations, leur moyenne et leur médiane par matière enseignée et globales sont à renseigner.

Pour la promotion, la moyenne et la médiane par matière enseignée et globales sont à renseigner ainsi que leurs valeurs minimum et maximum.

Par suite diverses statistiques sur les notes (moyenne et médiane notamment) sont à représenter graphiquement.

L'utilisation de **JfreeChart** est vivement recommandé.

## IV VERSION 4  : Mode graphique avec un jeu de tests

Les données sont codées en dur dans des classes de tests ou bien lues dans des fichiers au format csv.

Une version en mode graphique comprenant toutes les fonctionnalités précédentes (mode console, fichiers csv, statistiques) est à développer.

Le design de l'interface graphique est à soigner.

L'utilisation de la classe **JTable** est recommandé.