TD 3

Exercice 1 : Reprise du TD1

Récupérer la librairie de gestion de commandes du TD1. Note : il pourra aussi être nécessaire de récupérer la classe `Paire` (en la recopiant ou en la rajoutant comme dépendance à votre projet).

Question 1 : analyser la classe `DAO`. Pour chaque méthode utilisant l'API Stream (`produits`, `selectionCommande`, `selectionCommandeSurExistanceLigne`, `selectionProduits`), en donner une version ne l'utilisant pas. Discuter.

Question 2 : refactorer classe `Commande` à l'aide de l'API Stream. Pour cela :

- définir une function `formatteurLigne` permettant d'obtenir la représentation sous forme de chaîne de caractère d'une ligne de commande
- réécrire la méthode `toString` en utilisant `formatteurLigne`, `map` et `collect`
- faire en sorte que l'on puisse utiliser différents formatteurs de ligne pour différentes commandes (un même formatteur étant utilisé pour toutes les lignes d'une même commande), `formatteurLigne` étant utilisé par défaut si l'on ne précise rien
- réécrire la méthode `cout` avec `map` et `reduce`
- écrire une méthode générique `regrouper` permettant de regrouper des lignes (quelque soient les types dans les paires). Comparer à `Collectors::groupingBy`.
- réécrire la méthode `normaliser` en utilisant `regrouper`, `forEach` et `reduce`. Comparer à la version initiale. Discuter.

Question 3 : refactorer s'il y a lieu vos réponses à l'exercice 4 du TD1 et vérifier que les résultats sont les mêmes.

Exercice 2 : Reprise du TD2

Récupérer la librairie de gestion universitaire du TD2 (en la recopiant ou en la rajoutant comme dépendance à votre projet).

Question 1 : refactorer vos réponses à l'exercice 3 du TD2 en utilisant l'API Stream.

Vous écrirez pour cela les fonctions suivantes :

```
    // matières d'une année
    public static final Function<Annee, Stream<Matiere>> matieresA = ???

    // matières d'un étudiant
    public static final Function<Etudiant, Stream<Matiere>> matieresE = ???

    // matières coefficientées d'un étudiant (version Entry)
    public static final Function<Etudiant, Stream<Entry<Matiere, Integer>>>
matieresCoefE_ = ???
```

```
    // transformation d'une Entry en une Paire
    public static final Function<Entry<Matiere, Integer>, Paire<Matiere, Integer>>
entry2paire = ???

    // matières coefficientées d'un étudiant (version Paire)
    public static final Function<Etudiant, Stream<Paire<Matiere, Integer>>>
matieresCoefE = ???

    // accumulateur pour calcul de la moyenne
    // ((asomme, acoefs), (note, coef)) -> (asomme+note*coef, acoef+coef)
    public static final BinaryOperator<Paire<Double, Integer>> accumulateurMoyenne =
???

    // zero (valeur initiale pour l'accumulateur)
    public static final Paire<Double, Integer> zero = ???

    // obtention de la liste de (note, coef) pour les matières d'un étudiant
    // 1. obtenir les (matière, coef)s
    // 2. mapper pour obtenir les (note, coef)s, null pour la note si l'étudiant est
DEF dans cette matière
    public static final Function<Etudiant, List<Paire<Double, Integer>>>
notesPonderees = ???

    // obtention de la liste de (note, coef) pour les matières d'un étudiant
    // 1. obtenir les (matière, coef)s
    // 2. mapper pour obtenir les (note, coef)s, 0.0 pour la note si l'étudiant est
DEF dans cette matière
    public static final Function<Etudiant, List<Paire<Double, Integer>>>
notesPondereesIndicatives = ???

    // replie avec l'accumulateur spécifique
    public static final Function<List<Paire<Double, Integer>>, Paire<Double, Integer>>
reduit = ???

    // calcule la moyenne à partir d'un couple (somme pondérée, somme coefs)
    public static final Function<Paire<Double, Integer>, Double> divise = ???

    // calcul de moyenne fonctionnel
    // composer notesPonderees, reduit et divise
    // exception en cas de matière DEF
    public static final Function<Etudiant, Double> computeMoyenne = ???

    // calcul de moyenne fonctionnel
    // composer notesPondereesIndicatives, reduit et divise
    // pas d'exception en cas de matière DEF
    public static final Function<Etudiant, Double> computeMoyenneIndicative = ???

    // calcul de moyenne
    public static final Function<Etudiant, Double> moyenne = e -> (e == null || aDEF
.test(e)) ? null
```

```
            : computeMoyenne.apply(e);

    // calcul de moyenne indicative
    public static final Function<Etudiant, Double> moyenneIndicative =
computeMoyenneIndicative;
```