I) Etape 1 Collections/Comparable/Comparator

Récupérer le projet Maven **maven-collections-comparator** ou **app-collections-comparator** puis :

1) Compléter la classe **com.cours.entities.Personne** avec les attributs suivants :
« idPersonne» de type « Integer », « prenom » de type « String », « nom » de type « String », « poids» de type « Double », « taille » de type « Double », « rue» de type « String », « ville» de type « String », « codePostal» de type « String ».

Méthodes :

« toString () » qui retourne toutes les informations sur les attributs de la classe « Personne ».
« equals () » qui retourne un boolean true si les deux personnes ont le même prenom et le même nom.
« hashCode () » qui retourne un hash par rapport au prenom et nom.
« getImc () » qui retourne un double qui représente l'indice de masse corporelle.
« isMaigre () » qui retourne un boolean représentant l'état de maigreur de la personne.
« isSurPoids () » qui retourne un boolean représentant l'état de surpoids de la personne.
« isObese () » qui retourne un boolean représentant l'état d'obésité de la personne.
« compareTo (Personne other) » qui retourne un entier par rapport à une comparaison d'une personne avec idPersonne.

Indications :

Vous trouverez sur le lien ci-dessous toutes les informations pour les calculs des IMC :

https://fr.wikipedia.org/wiki/Indice_de_masse_corporelle

Dans tout l'exercice on initialisera les données avec les données ci-dessous.

1, Maurice, Dupont, 100, 170, rue du paradis, Rouen, 76000, France
2, Martin, Marshall, 55, 150, rue de Nantes, Laval, 53000, France
3, Claire, Chazal, 65, 175, rue de Rennes, Laval, 53000, France
4, Celine, Dia, 87, 170, rue Diderot, Paris, 75000, France
5, Remy, Cheval, 63, 140, rue du paradis, Nantes, 44000, France
6, Nicolas, Dutrou, 79, 155, rue Appert, Nantes, 44000, France
7, Marie, Claire, 65, 166, rue du paradis, Rouen, 76000, France
8, Nathalie, Sage, 89, 190, rue Appert, Rouen, 76000, France
9, Jean, Dujardin, 75, 140, rue des sorciers, Havre, 76800, France
10, Michel, Leclerc, 89, 190, rue du bonneur, Havre, 76800, France
11, Julien, Marshall, 60, 145, rue de Nantes, Laval, 53000, France
12, Julien, Claire, 78, 170, rue du Paradis, Paris, 75000, France
13, Jacques, Dupont, 63, 179, rue des Passeurs, Paris, 75000, France
14, Charles, Hallyday, 100, 200, rue des Feugrais, Rouen, 76000, France
15, Serge, Lama, 102, 195, rue des Heureux, Nantes, 44000, France
16, Vincent, Thomas, 81, 183, rue de Paris, Rennes, 35000, France
17, Eric, Dummat, 61, 155, rue de Versaille, Paris, 75000, France
18, Nicolas, Samuel, 64, 145, rue de Saint Louis, Laval, 53000, France

19, Rémy, Guerry, 79, 191, rue des Sages, Lyon, 69000, France
20, Nicolas, Drapeau, 56, 166, rue Mitterrand, Limoges, 87000, France

2) Utiliser la classe **ProcessCollections** et compléter la méthode **initArrayPersonnes** qui consiste à créer un tableau avec les 20 personnes ci-dessus, puis compléter **displayArrayPersonnes** qui consiste à afficher toutes les personnes.

   - Faire la même chose avec **initListPersonnes** qui consiste à créer une liste avec les 20 personnes ci-dessus, puis compléter **displayListPersonnes** qui consiste à afficher toutes les personnes.

   - Faire la même chose avec **initMapPersonnes** qui consiste à créer un dictionnaire (avec comme clés idPersonne et comme valeur Classe Personne) avec les 20 personnes ci-dessus, puis compléter **displayMapPersonnes** qui consiste à afficher toutes les personnes.

3) Compléter la classe **PersonneComparator** pour être en mesure de faire un tri des personnes d'une collections par **idPersonne, prenom et nom** (ascendante et descendante à chaque fois).

4) Utiliser la classe **ProcessComparator** et le comparateur **PersonneComparator** pour réaliser les méthodes **sortByIdAsc, sortByIdDesc, sortByPrenomAsc, sortByPrenomDesc, sortByNomAsc et sortByNomDesc** qui réaliseront les tries ascendants et descendants.

5) Faire des appels des classes **ProcessCollections** et **ProcessComparator** dans la classe **Main** pour vérifier que vos implémentations fonctionnent parfaitement.

II) Etape 2 Gestion personnes avec des stream

Récupérer le projet Maven **tp-peoples-streams** puis :

1) Créer l'interface **IPersonneDao** avec les méthodes

```
public interface IPersonneDao
{
   // obtenir tous les objets Personne
   List<Personne> findAll( );

   // chercher une Personne avec son id
   Personne findById( Integer id );

   // chercher une Personne avec son prenom
   List<Personne> findByPrenom( String prenom );

   // chercher une Personne avec son nom
   List<Personne> findByNom( String nom );

   // chercher des Personnes avec son prenom et son nom
   List<Personne> findByPrenomNom( String prenom, String nom );

   // chercher des personnes avec leur codePostal
   List<Personne> findByCodePostal( String codePostal );

   // chercher des personnes avec leur ville
   List<Personne> findByVille( String ville );

   // chercher les Personnes obese
   List<Personne> findPersonnesObese( );

   // chercher les Personne obese
   List<Personne> findPersonnesMaigre( );

   // obtenir tous les objets Personne
   Personne[] findAllArray( );

   // chercher une Personne avec son prenom
   Personne[] findByPrenomArray( String prenom );

   // chercher une Personne avec son nom
   Personne[] findByNomArray( String nom );

   // chercher des Personnes avec son prenom et son nom
```

```
    Personne[] findByPrenomNomArray( String prenom, String nom );

    // chercher des personnes avec leur codePostal
    Personne[] findByCodePostalArray( String codePostal );

    // chercher des personnes avec leur ville
    Personne[] findByVilleArray( String ville );

    // chercher les Personnes obese
    Personne[] findPersonnesObeseArray( );

    // chercher les Personne obese
    Personne[] findPersonnesMaigreArray( );

    // obtenir tous les objets Personne
    Map<Integer, Personne> findAllMap( );

    // chercher une Personne avec son prenom
    Map<String, Personne> findByPrenomMap( String prenom );

    // chercher une Personne avec son nom
    Map<String, Personne> findByNomMap( String nom );

    // chercher des Personnes avec son prenom et son nom
    Map<String, Personne> findByPrenomNomMap( String prenom, String nom );

    List<String> findAllPrenoms( );

    List<String> findAllNoms( );
}
```

2) Créer la classe **PersonneDao** qui implémentera l'interface **IPersonneDao**. On déclarera une liste de personnes avec lequel nous effectuerons les différentes opérations.

III) TP API DateTime

Créer le projet Maven Jar **tp-peoples-date-time**.

1) Créer la classe **com.cours.entities.Personne** avec les attributs suivants :
« idPersonne» de type « Integer », « prenom » de type « String », « nom » de type « String », « dateNaissance » de type « java.time.Instant», « poids» de type « Double », « taille » de type « Double », « rue» de type « String », « ville» de type « String », « codePostal» de type « String » et « pays» de type « String »

Méthodes :

« toString () » qui retourne toutes les informations sur les attributs de la classe « Personne ».
« equals () » qui retourne un boolean true si les deux personnes ont le même prenom et le même nom.
« hashCode () » qui retourne un hash par rapport au prenom et nom.
« getImc () » qui retourne un double qui représente l'indice de masse corporelle.
« isMaigre () » qui retourne un boolean représentant l'état de maigreur de la personne.
« isSurPoids () » qui retourne un boolean représentant l'état de surpoids de la personne.
« isObese () » qui retourne un boolean représentant l'état d'obésité de la personne.
« compareTo (Personne other) » qui retourne un entier par rapport à une comparaison d'une personne avec idPersonne.

Indications :

Vous trouverez sur le lien ci-dessous toutes les informations pour les calculs des IMC :

https://fr.wikipedia.org/wiki/Indice_de_masse_corporelle

Dans tout l'exercice on initialisera les données avec les données ci-dessous.
Créer le fichier personnesCsv.csv avec les données ci-dessous

idPersonne;Prenom;Nom;DateNaissance;Poids;Taille;Rue;Ville;Code Postal;Pays
1; Martin; Marshall;01-01-2000;60;150; rue de Nantes; Laval;53000;France
2; Claire;Chazal;01-01-2000;65;155; rue de Rennes; Laval;53000;France
3; Jacques; Dupont;01-01-2000;90;180; rue des Anges; Paris;75000;France
4; Celine; Dia;01-01-2000;66;166; rue Diderot; Paris;75000;France
5; Remy; Cheval;01-01-2000;88;200; rue du paradis; Nantes;44000;France
6; Nicolas; Dutrou;01-01-2000;40;150; rue Appert; Nantes;44000;France
7; Marie; Claire;01-01-2000;92;188; rue du paradis; Rouen;76000;France
8; Nathalie; Sage;01-01-2000;75;175; rue Appert; Rouen;76000;France
9; Jean; Dujardin;01-01-2000;56;166; rue des sorciers; Havre;76800;France

10; Michel; Leclerc;01-01-2000;100;202; rue du bonneur; Havre;76800;France
11; Julien; Marshall;01-01-2000;65;144; rue de Nantes; Laval;53000;France
12; Julien; Claire;01-01-2000;85;169; rue du Paradis; Paris;75000;France
13; Jacques; Dupont;01-01-2000;87;172; rue des Passeurs; Paris;75000;France
14; Charles; Hallyday;01-01-2000;69;133; rue des Feugrais; Rouen;76000;France
15; Serge; Lama;01-01-2000;78;174; rue des Heureux; Nantes;44000;France
16; Vincent; Thomas;01-01-2000;35;169; rue de Paris; Rennes;35000;France
17; Eric; Dummat;01-01-2000;56;155; rue de Versaille; Paris;75000;France
18; Nicolas; Samuel;01-01-2000;49;171; rue de Saint Louis; Laval;53000;France
19; Remy; Guerry;01-01-2000;54;157; rue des Sages; Lyon;69000;France
20; Nicolas; Drapeau;01-01-2000;63;145; rue Mitterrand; Limoges;87000;France
21; Maurice; Dupont;01-01-2000;100.0;170.0; rue du paradis; Rouen; 76000;France

2) Créer la classe **com.cours.main.MainApp** avec les méthodes suivantes :

- **processLoadPersonnesFromCsvFile** qui lit le fichier CSV et charge les données dans une List.

- Implémenter les méthodes **anonymousSortByPrenom** et **anonymousSortByNom** le trie de personnes par l'attribut « prenom » de l'animal et par l'attribut « nom » de la classe Personne.

- Implémenter les méthodes **generatePersonneFileSortedByPrenom** et **generatePersonneFileSortedByNom** pour générer les fichiers **personnes_by_first_name_csv.csv** qui contiendra la liste des personnes du fichier **personnesCsv.csv** à qui on a appliqué une trie par prenom et le fichier **personnes_by_last_name_csv.csv** qui contiendra la liste des personnes du fichier **personnesCsv.csv** à qui on a appliqué une trie par nom. Il est important que les fichiers **personnes_by_first_name_csv.csv** et **personnes_by_last_name_csv.csv** soit exactement au même format de données que le fichier **personnesCsv.csv** donc attention au format de date de naissances.

IV) TP Fork/Join Pool

Dans ce TP, nous allons apprendre à utiliser les méthodes asynchrones fournies par les classes **ForkJoinPool** et **ForkJoinTask** pour la gestion des tâches.

1) TP Folder Processor

Récupérer ou créer le projet squelette **tp-fork-join-files**.

Nous allons implémenter un programme qui recherchera les fichiers avec une extension spécifique dans des dossiers et ses sous-dossiers.

Afin d'arriver à votre objectif, on pourra dans cet TP :

- Créer la classe **com.training.folder.FolderProcessor** un **RecursiveTask** (qui hérite de **ForkJoinTask**) que nous permettra de trouver dans un dossier et ses sous dossiers les fichier d'une extension particulière.

- Créer la classe **com.training.main.Main** avec une méthode **processFolders** qui prendra en paramètre une liste de dossiers et une extension dans lesquelles nous feront les différentes recherches.

Aide : Chaque sous-dossier de ce dossier pourra envoyer une nouvelle tâche à la classe **ForkJoinPool** de manière asynchrone. Pour chaque fichier de ce dossier, la tâche vérifiera l'extension du fichier et l'ajoutera à la liste des résultats si elle continue.