POCA — TP noté n°2
Java streams

Jean-Baptiste Yunès
Jean-Baptiste.Yunes@u-paris.fr

2 décembre 2021

Rendu final

L'intégralité du code produit, les tests, etc doit être fourni à la date indiquée par l'enseignant. Il est important de ne fournir que le code source, et de prendre bien soin que rien ne manque. Le code sera fourni sous la forme d'une archive `tar.gz` ou `zip` (pas autre chose). Le code devra comprendre un fichier `README.txt` contenant le nom et le prénom (dans cet ordre !) de l'élève concerné. Les modalités de rendu seront donnés par l'enseignant mais se feront sur `moodle.u-paris.fr` dans le cours POCA et sous aucune autre forme.

TP n°2

Dans ce premier TP, on vise à utiliser le modèle de calcul en flot de données ainsi qu'à utiliser systématiquement le style fonctionnel de Java.

Préliminaire

Tout d'abord, il est demandé de créer une classe `Individu` ayant comme attributs : une `taille`, une `dateDeNaissance`, un `lieuDeNaissance`, un `poids` et une liste de `hobbys`. Ces attributs sont non modifiables après la construction de l'objet mais sont manipulables via des getters (`getTaille`, etc.).

Les lieux de naissance possible sont définis dans une énumération adéquate `LieuxDeNaissance` parmi les valeurs `TOKYO`, `PARIS`, `NAIROBI`, `LOS_ANGELES`, `BAMAKO`, `MOSCOU`, `CARACAS` et `TBILISSI`. Cette énumération devrait permettre l'affichage agréable du nom de la ville ainsi qu'obtenir le fuseau horaire correspondant.

Les dates de naissance devront utiliser le package `java.time` ($\geq$ Java 8), c'est-à-dire les dates Java 8 et être placées dans le fuseau horaire du lieu de naissance. Les dates

possibles seront entre le 1er janvier 1990 et le 31 décembre 2005 et à n'importe quelle heure de 0h00 à 23h59.

Les hobbys sont à prendre dans une énumération adéquate parmi les valeurs `JEUX_VIDEO`, `SPORT`, `RANDONNÉE`, `SKI`, `CINÉMA`, `CUISINE`, `LECTURE`.

Le poids doit être compris entre les valeurs 60kg et 100kg (au kg près). La taille sera comprise encore 150cm et 190cm (au centimètre près).

L'affichage des individus doit être « lisible », comme : `Né le ... à ..., mesure ...cm, pèse ...kg, aime ..., ..., ....`

Il est ensuite demandé de créer une classe `IndividuUtils` contenant deux méthodes statiques :

— `Individu createIndividu()` permettant d'obtenir un `Individu` dont les caractéristiques seront tirées au hasard conformément aux valeurs indiquées ci-dessus.
— `List<Individu> createIndividus(int n)` permettant d'obtenir une liste de n individus « tirés au hasard ».

1 Exercice

Écrire un test (classe `Test1` avec `main`) permettant, à l'aide de streams à partir d'une même liste de 100 individus tirés au hasard, de :

1. afficher tous les tous les individus nés en 2000 au mois de mars,
2. afficher les 5 premiers individus nés en juillet 1993,
3. afficher le nombre total d'individus nés en 1994,
4. afficher les 5 plus grands individus nés au mois de décembre de l'année 2001,
5. afficher les 5 plus petits individus de taille au moins 180cm nés en 2002,
6. afficher les 5 plus petits individus de taille au moins 180cm nés en 2002 par ordre décroissant de leur taille,
7. afficher la liste (sans redondance) des hobbys des 5 plus lourds parmi les 20 plus grands individus nés en 1999,
8. afficher la liste (sans redondance) des mois de naissance des 20 plus grands individus nés en 1997.

2 Exercice

Dans une classe `Test2` avec un `main` :

1. surcharger la méthode `IndividuUtils.createIndividus` de telle sorte que l'on puisse contrôler le type concret de la liste obtenue en résultat, tester avec `ArrayList` et `LinkedList`.

2. surcharger la méthode `IndividuUtils.createIndividus` de telle sorte que l'on puisse contrôler la source de création des individus, tester avec un appel à la méthode `IndividuUtils.createIndividu` ou un `Supplier<Individu>` renvoyant toujours un même individu.
3. créer une méthode `void test(List<Individu> l)` permettant d'afficher la liste des individus à l'aide d'un for étendu. Pour les questions de l'exercice 1, reprendre celles qui génèrent des listes d'individus et obtenir en résultat une liste, laquelle sera envoyée à la méthode `test` pour affichage.
4. surcharger la méthode `test` de sorte que l'on puisse contrôler l'action de « sortie ». Tester avec une sortie réalisant deux affichages : l'affichage de base suivi d'un affichage customisé de l'individu (par exemple en affichant que ses hobbys).
5. créer un `Consumer<Individu>` permettant de stocker des individus dans un fichier. Utiliser ce `Consumer` de sorte à stocker dans un fichier l'ensemble des individus de la liste initiale (appel à la méthode `test` surchargée).
6. créer un `Supplier<Individu>` permettant de lire les individus stockés dans un fichier. Utiliser ce `Supplier` pour « injecter » des individus dans la liste crée via `createIndividus` avant les tests.

3 Exercice [difficile]

Dans une classe `Test3` avec un `main` on souhaite obtenir à partir du stream des individus une `Map<int,Set<Individu»` où la clé est une année de naissance, et le `Set` un ensemble de maximum 3 individus, de sorte que :

— les clés du tableau associatif soient triées dans l'ordre naturel,
— l'ensemble contient les trois (au plus) plus grands individus de l'année correspondante et triés par taille décroissant.

4 Exercice [très difficile]

Essayer de recoder des streams de votre cru. La classe s'appellerait `MyStream<T>` et permettrait de construire un « stream » à l'aide d'itérateurs sur des `Collection`s. Sur ces streams on pourrait filtrer (`MyStream<T> filter(Predicate<T>)`) et transformer (`MyStream<U> map(Function<T,U>)`), avec une opération finale de type `void forEach(Consumer<T>)`.

Ajouter d'autres fonctionnalités au gré de vos envies...