Haute École Léonard de Vinci

Institut Paul Lambin — Session de janvier 2022

Examen de Programmation Java : avancé

Titulaire(s) : Stéphanie Ferneeuw, Laurent Leleux, Raphaël Baroni, Baptiste Lapierre

Année(s) d'études : Bloc 2

Durée de l'examen : 3 h ; pas de sortie durant les 60 premières minutes

Modalités : théorie et JAVADOC accessible

/!\ Lisez entièrement l'énoncé avant de démarrer l'examen /!\

Préliminaires indispensables au bon déroulement de l'examen

L'examen comporte 4 questions, chacune calibrée pour nécessiter 35 minutes de travail. Vous avez donc 40 minutes pour lire cet énoncé, vous imprégner du projet, remettre votre travail final…

Chaque question est indépendante et doit être réalisée dans son propre projet. Ceci implique que vous pouvez **répondre aux questions dans l'ordre que vous voulez** ! C'est la raison pour laquelle vous avez 4 projets dans le zip initial. Cependant, ces 4 projets sont tous identiques. Chaque question part de la même base de code.

1. L'archive que vous avez dézippée contient :
    - 4 projets sur lesquels vous allez travailler, un par question
    - Les slides de COO
    - Les fiches de théorie de AJ
    - Les solutions des fiches de l'année
    - Le manuel IntelliJ (au cas où)
    - L'énoncé que vous êtes en train de lire
2. La JavaDoc est disponible dans C:\Progs\Java\...
3. Ouvrez le projet dans IntelliJ ou importez le projet dans Eclipse
4. Dans Eclipse, ajoutez JUnit5 au build path : Configure build path -> Libraries -> classpath -> Add Library -> JUnit -> JUnit5 -> Finish
5. Configurez **si nécessaire** le JDK (uniquement s'il n'est pas trouvé)
    - Dans Eclipse : Configure build path -> Libraries -> Add Library -> JRE System Library...
    - Dans IntelliJ : File -> Project Structure -> Project -> SDK
6. Répondez aux questions dans l'ordre que vous voulez, toujours dans le bon projet
7. Une fois terminé, renommez vos projets finaux en y mettant votre nom et prénom, faites une archive ZIP contenant **UNIQUEMENT** vos 4 projets finaux et renommez-la en mettant votre nom et prénom également.

- Nom des projets : **exam-22-01-question1-NOM-PRENOM**
- Nom de l'archive : **exam-22-01-NOM-PRENOM**

8. L'archive doit être à la racine du lecteur réseau.
9. Vérifiez une dernière fois que votre archive contient bien vos projets modifiés
10. Ecrivez votre Nom et Prénom sur votre feuille de login, et **remettez-la au professeur**.

Introduction

Voici une petite définition de « Open Data » tirée de Wikipédia :

> *Les données ouvertes (en anglais : open data) sont des données numériques dont l'accès et l'usage sont laissés libres aux usagers, qui peuvent être d'origine privée mais surtout publique, produites notamment par une collectivité ou un établissement public. Elles sont diffusées de manière structurée selon une méthode et une licence ouverte garantissant leur libre accès et leur réutilisation par tous, sans restriction technique, juridique ou financière.*
>
> *L'accès aux données vise d'une part à permettre aux citoyens de mieux contrôler l'administration, d'autre part d'exploiter ces données, ce qui implique que ce droit d'accès s'accompagne d'un droit à la réutilisation des données.*

Dans notre projet, nous allons analyser des données de StatBel (L'office Belge de statistique), à savoir, les prénoms de la population totale par commune pour l'année 2021.

Ces données sont représentées simplement sous forme d'un fichier CSV (déjà dans les ressources de votre projet) qui contient une ligne par commune et par prénom. Pour chaque prénom de chaque commune, le nombre de prénoms est repris. Attention, les chiffres ne sont publiés qu'à partir de 5 occurrences. Dès lors, vous n'y trouverez pas les prénoms donnés qu'une seule fois par exemple. Ceci justifie que la somme de toutes les occurrences de tous les prénoms de toutes les communes ne soit pas égale à la population totale Belge.

Dans notre application, nous avons modélisé cela à l'aide de 2 classes : `Firstname` et `City`. Ces deux classes sont liées. Chaque prénom retient un dictionnaire des villes où ce nom a été donné, ainsi que le nombre de fois pour chaque ville. De même, une ville retient tous les prénoms qui y ont été donnés.

La classe `Client` va lire le fichier, et stocker les données dans 2 `Map`. Ne vous en souciez pas.

Ce projet permet actuellement de calculer quelques statistiques sur base de ces données brutes (valeurs des `Map`). Vous allez devoir les compléter, tester, refactorer le code…

Bon travail !

Question 1 : collections, énumérés, JUnit, Mock Object

Dans cette question, vous devrez travailler dans les classes `City`, `FirstName` et `Client`. Vous devrez également créer une classe de test JUnit afin de tester la méthode `getBirthsCount` de la classe `StreamStatsService`.

a) Les méthodes `getCounts` de la classe `FirstName` et `getFirstnames` de la classe `City` renvoient directement la collection, ce qui n'est pas conseillé car cela aura comme conséquence qu'il sera possible de modifier cette collection de l'extérieur. Modifiez ces méthodes afin qu'elles renvoient des collections qui ne pourront pas être modifiées (une `UnsupportedOperationException` sera lancée si on essais d'ajouter, … une donnée dans la collection renvoyée)

b) Ajoutez dans la classe `FirstName` une méthode renvoyant toutes les villes dans lesquelles ce prénom a été donné trié par ordre alphabétique du nom (name) de la ville et, en cas d'égalité de nom, sur le code postal (id).

c) Créez un énuméré `Gender` dans la classe `FirstName`. Cette énuméré contiendra les constantes F et M. Elle aura aussi un champ name qui prendra la valeur « féminin » pour la constante F et « masculin » pour la constante M. Faites-en sorte que la méthode `toString` renvoie ce champ.
Vous devez ensuite utiliser cet énuméré comme type du champ `gender` et adapter le constructeur, getter, setter et la classe `StreamStatsService` afin de tenir compte de ce changement.

d) Créez une classe de test JUnit pour la classe `StreamStatsService`. Vous devez y tester la méthode `getBirthsCount`. Créez pour cela au moins deux FirstName contenant plusieurs villes. Vérifiez que la valeur renvoyée par la méthode `getBirthsCount` est bien la bonne. Vous devez faire ce test de deux façons : une fois sans utiliser Mockito (utilisez directement les classes fournies) et une fois avec Mockito.

Question 2 : streams

Dans cette question, vous travaillerez uniquement dans la classe `StreamStatsService`. Lisez bien la javadoc ainsi que les TODOS pour comprendre ce qui est attendu.

Vous avez 6 méthodes à compléter dans cet exercice :

a) `GetFirstnamesStartingWith` : retourne tous les prénoms qui commencent par le préfixe donné en paramètre. La vérification doit être case-insensitive.

b) `GetLongestFirstname` : retourne le prénom le plus long.

c) `GetLongestFirstnames` : retourne les x prénoms les plus longs. S'il y a plusieurs prénoms de même longueur, la méthode prends les premiers pour avoir une liste de maximum x prénoms.

d) `cityWithMaxFemaleNames` : retourne la ville (`City`) qui contient le plus de prénom féminin différents.

e) `top5NamesStartingWithPrefix` : retourne la liste des 5 noms commençant par le préfixe passé en paramètre ayant le moins d'occurrences en Belgique, ou moins si moins de 5 prénoms commencent par ce préfixe.

f) `numberOfCitiesWithMoreMThanF` : retourne une `map` qui contient d'une part, sous la clé `true`, le nombre de ville qui possède plus de prénoms masculins différents que de prénoms féminins différent, et d'autre part, sous la clé `false`, le nombre de ville dans le cas inverse.

Attention ! Chacune de ses méthodes ne doit contenir qu'une seule instruction. Dit autrement, le stream ne doit faire qu'une seule ligne. Evidemment, vous pouvez indenter votre code sur plusieurs lignes pour qu'il soit plus lisible.

Question 3 : collections, threads & classes imbriquées

Pour cette question, vous devez seulement travailler dans la classe `MultiThreadedStatsService`.

Veuillez bien lire la JavaDoc ainsi que les « `// TODO` » donnés dans cette classe : tout y est indiqué.

Dans un premier temps, vous allez exécuter séquentiellement autant de calculs qu'il y a eu de prénoms répertoriés en Belgique en 2021 (12317 prénoms répertoriés). Vous allez calculer le nombre total d'occurrences du prénom en Belgique et utiliser une collection pour enregistrer tous les résultats afin de pouvoir rapidement trouver le prénom ayant le plus d'occurrences associées.
Suite aux calculs, vous allez afficher ces résultats :

- le nombre maximum d'occurrences en Belgique associées à un même prénom
- le ou les prénom(s) associés à ce nombre maximum
- le temps en millisecondes pour faire tous les calculs (c'est déjà fait pour vous ;)

Dans un second temps, vous allez exécutez les mêmes calculs, mais d'une manière différente. Vous allez essayer de réduire le temps de processing d'une façon assez simpliste : vous allez lancer tous les calculs en parallèle (threads) et affichez les mêmes résultats que précédemment une fois tous les calculs terminés.

Question 4 : Les interfaces, factory et injection de dépendances

Comme vous pouvez le constater, le package `domain` contient des classes du domaine dont l'implémentation ne devrait pas être visible aux autres packages.

Remédiez à ce problème en **plaçant des interfaces aux bons endroits, cachant les implémentations et en proposant une `factory` qui va distribuer les objets du domaine**.

Cette `factory`, ainsi que tous les autres objets de service devront être injectés par injection de dépendances à ceux qui en ont besoin. Le `main` s'occupera de cela et fera appel à une classe `InjectionService` que vous allez devoir compléter. Le rôle de cette classe est de :

- lire un fichier `dependencies.properties` qui définit le mapping entre les interfaces et les implémentations ;
- distribuer les dépendances à ceux qui en demandent.

N'oubliez pas de **compléter le fichier `dependencies.properties`** pour y ajouter les dépendances nécessaires.

Faites également **attention à l'encapsulation** (visibilité des classes/interfaces).