Formation : INFO2

INFO
INGÉNIEUR
ESIPE

Rapport du TD n°08
Java avancé

Guillaume CAU
27/09/2019

UNIVERSITÉ PARIS-EST
MARNE-LA-VALLÉE

Esipe
INGÉNIEURS PARIS EST

INGENIEURS
2000
L'EXCELLENCE AU SERVICE DE L'ALTERNANCE

I) Question 01 :

L'objectif de cette première question était de créer les fonctions principales de la classe Calc.

C'est à dire les fonctions :

- La méthode **set()** : Cette première méthode sert à insérer la valeur d'une case. Cette valeur est un **Supplier** qui renvoie la vrai valeur de la case ou qui la calcule.
  La méthode Set utilise simplement la méthode put de la classe HashMap utilisée dans l'objet interne de notre classe.
- La méthode **eval()** : Cette méthode appelle la fonction contenue dans le **Supplier** et renvoie la valeur de retour.
  Celle-ci appelle simplement la méthode get() de l'interface fonctionnelle stockée dans la case.

2) Question 02 :

L'objectif de cette question est de créer la méthode toString(). Pour ce faire, j'ai utilisé les Stream. Cette classe Stream récupère les valeurs des cases puis utilise le Collector.joining() pour faire la chaîne de caractères.

3) Question 03 :

L'objectif de cette question était de créer la méthode forEach(). Pour ce faire, j'ai simplement appliqué le BiConsumer à chaque élément de ma HashMap.

4) Question 04 :

L'objectif de cette question était de créer l'interface Group qui permet de représenter un groupe de case. Cette interface à la particularité d'être une **interface fonctionnelle**.

Les varargs peuvent poser plusieurs problèmes :

- On ne peut pas passer une liste facilement.
- Si le nombre d'arguments est trop importants, l'appel à cette méthode est plus compliqué.

5) Question 05 :

L'objectif de cette question était de créer la méthode forEach sur l'interface Group. Pour ce faire, il faut savoir qu'au moment de l'appel de la fonction, l'interface est implémentée. On peut donc appeler la méthode de l'interface fonctionnelle pour appeler la méthode forEach sur les éléments qu'elle renverra.
J'ai donc simplement appelé la méthode **values()** puis appliqué la méthode passé en arguments à chacun des éléments de la liste.

6) Question 06 :

L'objectif de cette question était de créer la méthode **matrix**. L'objectif de cette méthode était simplement de retourner les numéros de cases entre certaines bornes.
J'ai donc simplement utilisé les stream renvoyées par IntStream pour générer les noms de cases correspondante.

7) Question 07 :

L'objectif de cette question était de créer la méthode ignore. Cette méthode n'a pas trop été difficile à développer. J'ai simplement utilisé la méthode **filter()** de Stream.

8) Question 08 :

Cette question a été pour moi, la plus difficile du TP. En effet, trouver son prototype m'a pris beaucoup de temps.
L'objectif de la méthode était de fournir une fonction qui permet de renvoyer une Stream contenant tout les éléments calculés de la Stream.
Une fois le prototype trouvé, il m'a suffit de filtrer les Optional vide puis de renvoyer le résultat.

Conclusion :

J'ai beaucoup apprécié ce TP. Il m'a permis d'appliquer mes connaissances en Lambda et en interfaces fonctionnelles dans un cas concret : la création d'un tableur.
Avant ce TP, j'avais du mal à trouver un cas dans lequel j'allais pouvoir créer une interface fonctionnelle et ce TP m'a permis d'en découvrir un.