COMPTE RENDU : MARGHT ABDE-RRAHMAN ET BENNOURA BOUCHIBA KARIM

Dans ce projet, nous avons implémenté un calculateur d'empreinte carbone capable de quantifier les émissions à effet de serre d'un utilisateur et de les lui indiquer sur la console.

Par souci de simplification du problème, seuls les postes de consommation carbone suivants ont été considérés : l'alimentation, le(s) logement(s), le(s) véhicule(s), les biens de consommations, les services publics, et un poste de notre choix, les appareils numériques.

Par manque de temps, nous n'avons pas réalisé d'interface graphique, un utilisateur de notre calculateur sera donc amené à l'utiliser par le biais de notre menu interactif et devra rentrer ses données directement sur la console. L'empreinte carbone de l'utilisateur lui sera alors détaillée sur la console, et des conseils lui seront délivrés.

STRUCTURE DU PROJET ET DIFFICULTES RENCONTREES :

A/ Package consoCarbone:

Pour réaliser le calculateur d'empreinte carbone, nous avons implémenté une classe mère consoCarbone qui représente un poste de consommation carbone de base. Elle possède un attribut impact qui représente l'impact carbone du poste de consommation considéré. Puis, nous avons implémenté l'ensemble des classes correspondant à chaque poste de consommation soit : Logement, Transport, BienConso, ServicesPublic , Alimentation et Numerique. Étant donné que chacune de ces classes possède l'attribut impact et représente un poste de consommation,il a été pertinent de les faire hériter de la classe ConsoCarbone qui est alors une classe abstraite.

Dans chacune des classes nous avons redéfini la méthode toString() pour qu'elle affiche un message clair quant à la valeur de l'empreinte carbone liée au poste de consommation de la classe.

Une fois toutes ces classes implémentées, nous avons commencé l'écriture du menu interactif. On y trouve une méthode main qui est le corps principal du déroulement du menu interactif. L'utilisateur peut choisir le poste de consommation duquel il veut que lui soit présenté son empreinte carbone et tant qu'il ne choisit pas de quitter le menu interactif, il peut demander à calculer l'empreinte carbone d'un autre poste de consommation dont il devra indiquer sur la console les informations. Ceci est écrit grâce à un switch dans une boucle do/while.

Pour alléger le corps de la méthode main, nous avons choisi d'écrire une méthode de récupération d'information et d'affichage pour chacun des postes de consommation.

On a tout d'abord écrit l'interaction avec l'utilisateur dans chacune des méthodes sans se soucier des exceptions, mais nous avons tout de même rencontré une erreur. Nous initialis**ions** un objet de la classe Scanner dans chacune des méthodes sans le fermer à chaque fois. Nous avons réglé le problème en initialisant un seul object Scanner au début de notre méthode main puis nous avons donné en paramètre de chaque méthode d'affichage l'objet scanner.

Après écriture de l'interaction, nous nous sommes occupés des exceptions. Il nous a fallu créer une classe d'exceptions pour chacun des postes de consommations et chaque méthode d'affichage utilise l'exception correspondante à son poste de consommation. Ce travail a été assez répétitif. De même pour le traitement des exceptions, c'est surtout le début qui a été difficile à implémenter.

Nous avons commencé par l'écriture de affichage_alimentation() dans lequel les exceptions traitées ont été les suivantes: les taux de repas végétariens et à base de boeuf devaient être compris entre 0 et 1, la somme des deux devaient être également comprise entre 0 et 1 et enfin l'utilisateur devait entrer sur la console un chiffre de sorte que l'on puisse traduire la chaîne de caractères en double. Étant donné que nous demandons à d'autres reprises dans les autres fonctions d'affichage l'entrée sur la console d'un chiffre par l'utilisateur, il nous a suffi de copier/coller un bout de code et de l'adapter à l'exception traitée.

En lançant et en essayant le menu interactif nous nous sommes rendus compte de quelques soucis de clarté que l'on a pu améliorer. Par exemple, lorsque le taux de repas à base de bœuf est une valeur qui

n'est pas comprise entre 0 et 1 nous redemandons seulement ce taux à l'utilisateur et non les deux taux comme c'était le cas dans notre code initial. De même lorsque l'utilisateur entrait une valeur inappropriée sur la console lorsqu'on lui demandait s'il possédait une tablette par exemple, nous lui posions à nouveau l'ensemble des questions quant à la possession d'appareils numériques au lieu de lui demander seulement une nouvelle fois s'il possédait une tablette. Ceci venait du fait que nous avions écrit une seule grande boucle do/while au lieu d'en écrire une pour chacune des questions posées et donc pour chacune des exceptions.

Pour la récupération des informations et l'affichage de l'empreinte carbone d'objets de la classe Logement et de la classe Transport nous avons adopté une même logique. Nous avons jugé pertinent de créer deux méthodes, une qui sera appelée dans la méthode main et qui va demander à l'utilisateur combien de logements ou de véhicules il possède, et une qui va s'occuper de récupérer les informations de chacun de ses logements/véhicules. Cela a permis de clarifier l'écriture de notre code.

La documentation javadoc a été relativement fastidieuse, nous avons fait en sorte de la réaliser au fur et à mesure de notre avancement dans le projet avant de la compléter au mieux et de nous relire une fois le code terminé. L'écriture des classes du package consoCarbone n'a pas particulièrement posé de difficultés, seul la classe Menu a demandé beaucoup de temps d'écriture et de débugage. Nous avons réalisé la classe Numérique à l'aide des documents fournis au début du sujet. Une fois les classes du package consoCarbone écrite, nous avons pu passer à l'écriture dans un nouveau package d'une classe Utilisateur regroupant l'information de tous les postes de consommation carbone.

B/Package Suite:

Nous avons implémenté une classe "Utilisateur" qui répertorie tous les postes de consommation d'un individu, dans laquelle nous avons programmé 3 constructeurs différents. Un par défaut, un autre qui prend en paramètre les postes de consommations directement et, un dernier, qui prend le nom d'un fichier texte (dans notre cas : "MesInfos.txt") qui contient les informations nécessaires à la création d'un utilisateur et de ces postes de consommations. ATTENTION au format du fichier texte.

D'ailleurs, pour ce constructeur, nous avons eu un bon lot de difficultés pour récupérer les informations, pour régler cela j'ai eu à utiliser un compteur "cpt", pour faire attention à quelles informations nous récupérons. Nous forçons l'utilisateur à écrire les informations de ses postes de consommations carbones à des endroits précis du fichier pour pouvoir les récupérer plus facilement étant donné que nous savons donc à quelle ligne est supposé se trouver l'information sur le fichier modifié par l'utilisateur.

Nous n'avons pas eu le temps de nous occuper des exceptions qui peuvent être levées dans cette méthode. Il nous aurait par exemple fallu vérifier que l'utilisateur ait bien écrit un chiffre là où on attend un chiffre sans quoi il aurait fallu lui proposer de modifier le document avant de relire et de vérifier une nouvelle fois si l'utilisateur ne s'est pas trompé dans quel cas on pourrait alors continuer de récupérer les informations sur le fichier texte.

Nous avons, ensuite, implémenté une classe "Population" qui répertorie un ensemble d' "Utilisateur". Dans cette classe, on calcule la moyenne de chaque poste de consommation de la population avec la méthode "moyenneImpact". Lorsque la moyenne d'un poste de consommation est supérieure à la moyenne des français, notre programme affiche des conseils pour réduire la consommation carbone.

C/Package Tests:

Dans le package “test”, comme vous pouvez le voir nous avons trois classes. La première, “TestConsoCarbone”, nous a permis de faire quelques tests unitaires sur la classe abstraite “ConsoCarbone”. Avant chaque test, on a décidé d’initialiser un poste de consommation de classe “Logement” de superficie 100 et de classe énergétique C, sur lequel nous avons fait des tests de comparaison pour vérifier notre méthode “compareTo()” .

La seconde, “TransportTest”, nous a permis de faire quelques tests unitaires sur la classe “Transport”, qui hérite de “ConsoCarbone”. On commence en initialisant 4 objets de la classe “Transport” qui nous ont permis de faire des tests sur des “Transport” différents.

On a essayé de tester un maximum de nos méthodes et nous sommes arrivés à un “coverage” de 78,4% pour la classe “Transport”. Malheureusement, nous n’avons pas réussi à arriver à 100% de “coverage”, car nous vérifions que les méthodes qui ont une variable de retour et pas de l’affichage.

La troisième, “TestUtilisateur”, nous a permis de faire quelques tests unitaires sur la classe “Utilisateur”. Dans cette classe de tests, avant chaque test, on initialise un utilisateur de consommations arbitraires. On a calculé, à la main, la consommation de cet utilisateur et avons vérifié que le calcul de notre méthode est bon.

Nous voulons préciser que par manque de temps, nous n’avons pas inclut la classe Population dans notre menu interactif et nos méthodes de politique public ne pourront donc pas être affichée sur la console.

Nous avons trouvé le sujet très intéressant, il nous a permis de nous documenter sur différents postes de consommation carbone et a été un véritable défi d’organisation qu’on aurait aimé mieux relever. Nous avons essayé d’être le plus clair possible lors de l’interaction sur le menu pour que l’utilisateur puisse profiter de la meilleure expérience avec notre calculateur d’empreinte carbone.