**Compte-rendu du projet**

Nous avons décidé de nous réunir tous les mercredis pour faire ce qui nous a été demandé par semaine. De cette manière, nous avons pu être dans les temps et ne pas se sentir submergé par le travail. Pour chaque cours, on a échangé nos idées pour prendre une décision qui satisfera tout le monde. Lors de l’avancement du projet, il est évident qu’on a dû revenir en arrière de nombreuses fois afin d’améliorer notre programme.

Ce compte-rendu est divisé en trois parties. Nous présenterons dans un premier temps la structure de notre code. Puis, nous expliquerons les extensions que nous avons choisi pour notre projet. Enfin, nous mettrons en avant certains problèmes auxquels nous avons été confrontés.

I. Présentation de la structure du code :

Package ConsoCarbone :

- Classe **ConsoCarbone** : Cette classe représente un poste de consommation générique.

Elle implémente deux interfaces :

→ l’interface **Comparable** afin de pouvoir comparer deux postes de consommations à l’aide de la fonction **compareTo**

→ l’interface **Serializable** afin de pouvoir lire et écrire un objet de la classe dans un fichier (utile pour la partie entrée/sortie).

Nous avons fait le choix de déclarer l’attribut **impact** dans cette classe et de définir le getter et le setter de cet attribut afin de le récupérer ou de le modifier dans les classes filles. Cela nous a évité de les redéfinir dans chacune des classes filles.

On y trouve également l’attribut **id** qui est

→ **protected** afin de pouvoir y accéder dans chaque classe fille (on peut également le récupérer à l’aide de **getId)**

→ **final** qui signifie qu’on ne pourra pas le modifié.

L’attribut **cpt** est **static** car sa valeur est identique pour tous les objets de la classe.

- Classe **Alimentation** : Cette classe est une classe fille de la classe **ConsoCarbone** car elle représente le poste de consommation concernant l’alimentation.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

On a l’attribut **txBoeuf** qui correspond au taux de repas à base de bœuf et l’attribut **txVege** qui correspond au taux de repas végétariens. Nous avons ajouté des exceptions dans leur setter (**setTxBoeuf** et **setTxVege**) ainsi que dans les constructeurs afin de s’assurer que leur valeur soient bien entre 0 et 1.

- Classe **Logement** : Cette classe est une classe fille de la classe **ConsoCarbone** car elle représente le poste de consommation concernant le logement.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

On y retrouve deux attributs : **superficie** qui correspond à la superficie en m^2 du logement et **ce** de type **CE** qui correspond à la classe énergétique du logement (A, B, C, D, E, F et G). Nous avons ajouté une exception dans **setSuperficie** et dans les constructeurs afin de s’assurer que la superficie du logement soit positive.

- Énumération **CE** : Il s’agit d’une énumération comportant les sept différentes classes énergétiques d’un logement (A, B, C, D, E, F et G). Chacune est associée à un coefficient multiplicatif noté **alphaCE** nécessaire pour calculer l’impact du logement. On le récupère à l’aide de la méthode **getAlphaCE**.

On a également la méthode **value** qui permet de convertir une chaîne de caractère en objet CE, la chaîne de caractères prises en argument doit nécessairement valoir A, B, C, D, E, F ou G.

- Classe **BienConso** : Cette classe est une classe fille de la classe **ConsoCarbone** car elle représente le poste de consommation concernant les dépenses en biens de consommation.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

On a l’attribut **montant** qui correspond au montant des dépenses annuelles de l’utilisateur. Nous avons ajouté une exception dans **setMontant** et dans les constructeurs afin de s’assurer que le montant soit positif. Comme une tonne de CO2eq est équivalente à 1750€ de dépenses, nous avons divisé le montant par 1750 pour le calcul de l’impact.

- Classe **ServicesPublics** : Cette classe est une classe fille de la classe **ConsoCarbone** car elle représente le poste de consommation concernant l’utilisation des services publics.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

Comme tous les français ont la même empreinte carbone pour ce poste de consommation, nous avons décidé d’utiliser le design pattern Singleton qui permet de s’assurer que la classe ne possède qu’une seule instance.

- Classe **Transport** : Cette classe est une classe fille de la classe **ConsoCarbone** car elle représente le poste de consommation concernant les déplacements.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

Nous avons décidé qu’elle devait être abstraite car elle possède les méthodes abstraites **setDistance** et **getDistance** qui sont à redéfinir dans les classes filles **Voiture**, **Avions** et **TrainEtBus**. Elle contient donc l’attribut **distance** (distance parcourue en une année) qui est **protected** afin de pouvoir y accéder dans les classes filles et d’éviter de le redéfinir dans chacune d’entre elles.

- Classe **Voiture** : Cette classe est une classe fille de la classe **Transport** car elle représente les déplacements en voiture.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

On a trois attributs :

→ l’attribut **possede** qui vaut true si l’utilisateur possède une voiture et false sinon

→ l’attribut **taille** de type Taille qui permet de savoir s’il s’agit d’une petite ou d’une grosse voiture

→ l’attribut **amortissement** qui correspond à la durée de conservation du véhicule.

On y retrouve bien l’implémentation des deux méthodes **getDistance** et **setDistance** définie dans la classe **Transport**.

Nous avons ajouté une exception dans les setters **setAmortissement** et **setDistance** ainsi que dans les constructeurs afin de s’assurer que l’amortissement et la distance soient positifs.

- Énumération **Taille** : Il s’agit d’une énumération contenant deux instances P (pour petite voiture) et G (pour grosse voiture). Chacune est associée à une valeur noté **fabrication** qui correspond à l’émission nécessaire à la fabrication de la voiture. On la récupère à l’aide de la méthode **getFabrication**.

- Classe **Avion** : Cette classe est une classe fille de la classe **Transport** car elle représente les déplacements en avion.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

On y retrouve bien l’implémentation des deux méthodes **getDistance** et **setDistance** définie dans la classe **Transport**.

Nous avons ajouté une exception dans **setDistance** ansi que dans les constructeurs afin de s’assurer que la distance soit positive.

- Classe **TrainEtBus** : Cette classe est une classe fille de la classe **Transport** car elle représente les déplacements en train et en bus.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

On y retrouve bien l’implémentation des deux méthodes **getDistance** et **setDistance** définie dans la classe **Transport**.

Nous avons ajouté une exception dans **setDistance** ansi que dans les constructeurs afin de s’assurer que la distance soit positive.

Pour chacune des classes filles de la classe **ConsoCarbone**, nous utilisons à chaque fois la méthode **setImpact** définie dans la classe mère **ConsoCarbone** pour pouvoir calculer ou mettre à jour l’impact selon la formule donnée dans l’énoncé.

Dans ce package, nous retrouvons également les deux classes de test : **TestAlimentation** et **TestBienConso** qui permettent de tester certaines méthodes des classes **Alimentation** et **BienConso**.

Nous avons enfin créé la classe **ErrVal** pour pouvoir gérer les exceptions et personnaliser le message qui s’affiche pour les différentes exceptions.

Package Utilisateur :

- Classe **Utilisateur**: Cette classe permet de s’intéresser à la consommation carbone d’un utilisateur.

Elle implémente l’interface **Serializable** pour les mêmes raisons que la classe **ConsoCarbone**.

Elle possède les attributs **alimentation**, **bienConso**, **logements**, **avion**, **voiture**, **trainEtBus** et **services** qui correspondent aux différents postes de consommation. Seul l’attribut **logements** est une liste afin que l’utilisateur puisse avoir plusieurs logements. Nous avons réalisé les getters et les setters pour chacun des attributs et nous avons ajouté dans chaque setter ainsi que dans les constructeurs un bloc **try catch** qui permet de traiter les différentes exceptions.

Nous avons pris l’initiative de rajouter :

→ un attribut **id** afin de pouvoir identifier les différents utilisateurs, ce qui a été utile dans la classe **Population**.

→ une méthode **toString** afin de pouvoir afficher toutes les informations concernant un utilisateur.

On y trouve également la méthode **sortConsoCarbone** qui crée une liste d’objets **ConsoCarbone** et y ajoute les différents attributs correspondant aux différents postes de consommations. Puis, trie la liste selon les valeurs des impacts et présente dans l’ordre le résultat à l’utilisateur. Pour finir, on compare la valeur de chacun des impacts à l’empreinte carbone moyenne des Français concernant ce poste. Si c’est supérieur, on fait alors des recommandations à l’utilisateur pour qu’il obtienne un mode de vie plus durable.

- Classe **Population** : Cette classe permet de créer une liste d’utilisateur.

On y trouve donc l’attribut **population** qui est une liste d’utilisateur. Nous avons décidé d’implémenter quatre méthodes :

→ **addUtilisateur** qui prend en argument un utilisateur et le rajoute dans la liste

→ **display** qui prend en argument un entier et affiche les informations de l’utilisateur ayant cet entier comme id

→ **toString** qui permet d’afficher les informations concernant tous les utilisateurs de la population

→ **removeUtilisateur** qui prend en argument un entier et supprime de la liste l’utilisateur ayant cet entier comme id.

- Classe **Menu** : Cette classe est un menu interactif permettant d’interagir dans la console avec l’utilisateur du programme.

Nous avons quatre méthodes :

→ **printMenu** qui affiche le menu sur la console. Il permet à l’utilisateur de choisir entre trois actions : sortir du menu, écrire un Utilisateur dans un fichier ou encore lire un Utilisateur dans un fichier.

→ **ecrireUtilisateur** qui permet de demander toutes les informations nécessaires sur l’utilisateur afin de créer un objet de la classe Utilisateur (methode **creerUtilisateur**), et écrit celui-ci dans un fichier dont le nom est donné par l’utilisateur

→ **lireUtilisateur** qui permet de lire un objet de la classe Utilisateur contenu dans un fichier (dont le nom est donné par l’utilisateur) et de l’afficher sur la console

→ **creerPopulation** qui demande toutes les informations nécessaires sur une population à l’utilisateur puis affiche sur la console (selon les demandes de l’utilisateur) l’ensemble de ces informations mais également les mesures de politique publique à mettre en place afin de réduire l’empreinte carbone de cette population.

Cette classe contient également une méthode **main** qui demande à l’utilisateur quelle action il souhaite effectuer en appelant la méthode **printMenu**. On récupère ensuite sa réponse dans une variable **choice**. Selon la valeur de cette variable, on appelle les différentes méthodes à l’aide d’un bloc **switch** : **ecrireUtilisateur** si l’utilisateur a tapé 1, **lireUtilisateur** si l’utilisateur a tapé 2, **creerPopulation** si l’utilisateur a tapé 3 ou alors on sort du menu si l’utilisateur a tapé 0.

II. Présentation des extensions :

Calcul de l’impact dans les classes **Avions** et **TrainEtBus** :

- Dans la classe **Avion** : Nous avons utilisé le site suivant : <https://impactco2.fr/transport/avion>.

Nous n’avons pris en compte que le nombre de kilomètre parcourue afin de faciliter les calculs.

Ainsi, d’après les données de ce site :

→ lorsque le nombre de kilomètre parcourue est inférieur à 1000km, l’empreinte carbone par personne est de 230 gCO2eq/km ce qui fait 0.00023 TCO2eq/km et il suffit donc de multiplier la distance parcourue par l’utilisateur par 0.00023 pour obtenir l’impact de ses déplacements en avion

→ lorsque le nombre de kilomètre parcourue est compris entre 1000 et 3500km, l’empreinte carbone par personne est de 178,4 gCO2eq/km ce qui fait 0.0001784 TCO2eq/km et il suffit donc de multiplier la distance parcourue par l’utilisateur par 0.0001784 pour obtenir l’impact

→ lorsque le nombre de kilomètre parcourue est supérieur à 3500km, l’empreinte carbone par personne est de 151,66 gCO2eq/km ce qui fait 0.00015166 TCO2eq/km et il suffit donc de multiplier la distance parcourue par l’utilisateur par 0.00015166 pour obtenir l’impact.

- Dans la classe **TrainEtBus** : Nous avons utilisé le site suivant : <https://impactco2.fr/transport/busgnv>.

Nous n’avons pris en compte que le nombre de kilomètre parcourue afin de faciliter les calculs.

Ainsi, d’après les données de ce site, l’empreinte carbone par personne est de 121,90 gCO2eq/km ce qui fait 0.0001219 TCO2eq/km et il suffit donc de multiplier la distance parcourue par l’utilisateur par 0.00023 pour obtenir l’impact de ses déplacements en train et en bus.

Explication de la méthode **mesuresPolitiques** de la classe **Population** :

Dans cette méthode, nous avons fait le choix de nous intéresser à l’impact des déplacements en voiture des utilisateurs de la population ainsi qu’à la classe énergétique de leurs différents logements.

Tout d’abord, nous avons calculé l’impact moyen des déplacements en voiture de la population. Puis, nous vérifions si celui-ci est supérieur ou inférieur à l’empreinte carbone moyenne des français concernant les déplacements en voiture. S’il est supérieur, nous proposons alors un dimanche sans voiture afin de réduire l’impact des déplacements en voiture de la population.

Dans un second temps, nous avons créé une liste de CEcontenant toutes les classes énergétiques des différents logements des utilisateurs de la population. Puis, nous vérifions s’il existe des logements ayant une classe énergétique E, F ou G. Dans ce cas, nous proposons alors une rénovation énergétique de ces logements afin de passer à une meilleure classe énergétique et ainsi de réduire l’impact concernant le logement.

III.Difficultés rencontrées :

Lors de ce projet, nous avons rencontrés quelques difficultés. Tout d’abord, lorsque nous avons créé les classes **TestAlimentation**et **TestBienConso**on a eu du mal à les compiler. Cela provenait du fait que JUnit de fait pas partie de la bibliothèque standard de Java. Il nous a fallu du temps pour réussir à rajouter la bibliothèque JUnit.

Par ailleurs, lorsqu’on nous demandait de faire certaines choses qui nécessitaient des recherches comme étendre notre hiérarchie de classes du package **ConsoCarbone**. Il a fallu se documenter et souvent internet ne répondait pas à nos questions. Par exemple, pour l’impact de consommation carbone d’un avion, c’est à l’aide de plusieurs calculs fait à partir de différentes informations trouvées sur internet que l’on a pu retrouver ce qu’on cherchait.

Ce projet fut l’objet d’un long travail mais aussi d’un aboutissement. Au départ, nous avons pris peur face à ce qui été attendu mais après avoir analysé le sujet on a réussi à avancer pas à pas. Grâce à cela, nous nous sommes améliorées et ressentons une réelle progression entre le début et la fin du projet. Nous avons vu cela comme un challenge et nous en avons apprécié tout le processus. Cela nous a permis de nous dépasser en devant rendre un aussi grand travail dans les délais imposés.