Projet Empreinte Carbone

Groupe 11: Nada BORDJI, Amine LALLALI, Adel LAKHLEF

Ce rapport rend compte de la démarche suivie dans la création d'un programme capable de quantifier les émissions de gaz à effet de serre d'un(e) individu(e) en tonnes de CO2. Les données collectées se rapportent notamment au mode de vie de l'individu(e), à son alimentation, son logement, son moyen de transport... A partir de ces données, le programme est en mesure de conseiller l'individu(e) pour limiter son empreinte carbone.

Package consoCarbone :

Le package consoCarbone permet à l'aide de différentes classes le calcul de l'impact en termes d'émission de GES. Les classes contenues dans ce package permettent ce calcul en se basant sur des informations concernant leur domaine.

Nous avons la classe Alimentation qui demande le taux de repas à base de viande de bœuf et le taux de repas végétariens. Aussi on retrouve la classe Bienconso s'appuyant sur le montant des dépenses annuelles de l'utilisateur(rice). La classe CE est une énumération nous permettant de recueillir les coefficients multiplicatifs dépendants de la classe énergétique d'un logement. D'ailleurs ce coefficient est utilisé par la classe Logement en plus de la superficie de l'habitation. On retrouve une autre énumération, Taille afin de différencier la taille des voitures.
En addition avec le nombre de kilomètre parcourus en un an et la durée de conservation du véhicule, la classe Transport donne l'impact du transport de l'individu(e).
On retrouve aussi la classe ServicesPublics se chargeant de l'impact des services publics avec des valeurs statiques.

Enfin, la classe mère de toutes ces classes est la classe abstraite ConsoCarbone représentant un poste de consommation carbone générique.
Ces classes contiennent chacune des constructeurs, des getters et setters de chaque attribut. On retrouve souvent dans ces classes la méthode toString redéfinit pour permettre de retourner les attributs de la classe en chaines de caractères. Aussi la méthode français statique détaillant sur la console l'empreinte carbone moyenne d'un.e français.e vis-à-vis d'une classe.
Enfin la méthode compareTo est aussi présente dans la majorité de ces classes pour comparer les impacts issus des différentes classes.

Menu interactif + lecture fichier :

Le menu interactif est un outil d'une grande importance au traitement des données de l'utilisateur. En effet, le calculateur d'empreinte carbone a besoin d'informations entrées par l'utilisateur lui-même afin de procéder au calcul. La méthode permettant de lancer le menu interactif est **MenuInteractif()**, celle-ci prend aucun argument. Le menu offre différents modes de traitement de données.

Le premier est l'entrée de données par l'utilisateur lui-même. À l'aide de la classe *Scanner*, nous pouvons récupérer le flux d'entrée du clavier, ceci est réalisée par l'intermédiaire d'un objet du type *InputStream*, appelé *System.in*. Nous posons ainsi plusieurs questions auxquelles l'utilisateur devra y répondre afin de passer au calcul de son empreinte carbone. On a bien sûr pensé au cas où l'utilisateur se trompe lors de la saisie de données.

Pour gérer les erreurs, nous utilisons une boucle **While()**, et un bloc **Try{} Catch{}**, afin de traiter les erreurs et redemander la bonne valeur. Tant que l'utilisateur n'entre pas la valeur, un message d'erreur s'affichera sur la console.

Déclaration du scanner :

```
10          Scanner scanner = new Scanner(System.in);
```

Exemple d'utilisation du scanner:

```
System.out.println(x: "Voulez-vous lire un fichier ? Entrer Oui pour lire un fichier, Non sinon.");
boolean lirefichier = false;
succes = false;
while(!succes){
    try {
        entreeUt = scanner.next();
        if (!Arrays.asList(...a: "Oui","Non","oui","non").contains(entreeUt)) {
            throw new InvalidBooleanException();
        }
        if (entreeUt.equals(anObject: "Oui") || entreeUt.equals(anObject: "oui")) lirefichier = true;
        else lirefichier = false;
        succes = true;
    }
    catch (InvalidBooleanException e) {
        System.out.println(e.getMessage());
    }
}
```

Afin de récupérer la valeur entrée par l'utilisateur, nous utilisons une méthode de la classe *Scanner* qui est **next()**. Cette dernière retourne un *String* qui correspond à l'élément entré. On le stocke dans la variable **entreeUt**, puis on le traite en fonction des cas où nous nous situons (convertir en *Int*, convertir en *Boolean* ou le garder en *String* …).

Ici, l'utilisateur doit entrer *Oui* ou *Non* afin de lire un fichier texte. On voit bien le traitement d'erreur avec l'exception **InvalidBooleanException()** …

Exemple d'erreur :

```
Vous avez le choix entre lire un fichier contenant vos informations ou bien entrer les données vous même.
Voulez-vous lire un fichier ? Entrer Oui pour lire un fichier, Non sinon.
Ouicevzh

Réponse invalide. Si vous possédez une voiture entrez Oui, sinon entrez Non.
Non1

Réponse invalide. Si vous possédez une voiture entrez Oui, sinon entrez Non.
Nonagef

Réponse invalide. Si vous possédez une voiture entrez Oui, sinon entrez Non.
```

Tout ce procédé afin de récupérer les informations de l'utilisateur, est mis en place par la méthode **creeUtilisateurs().** Cette dernière prend en argument un *Scanner*, une *Collection d'Utilisateur* et le *nbUtilisateur.* Cette méthode est principalement composée de blocs similaires à

l'exemple ci-dessus traitant différents types de données (Superficie du logement, classe énergétique du logement, possède ou non un véhicule…). Une fois toutes les valeurs entrées on ajoute l'utilisateur créé en fin de boucle à la *Collection d'Utilisateur* entrée en argument afin de créer une *Population*.

Et cette fonction s'exécute *nbUtilisateur* fois afin de récupérer les données de tous les utilisateurs qui veulent entrer leurs données.

Cependant, en testant notre programme un problème est survenu. En effet, en entrant plus d'un utilisateur on remarque qu'à l'affichage de la fiche récapitulative de chaque individu, on obtient les données du dernier utilisateur uniquement. Nous avons essayé de résoudre le problème en se documentant, mais nous n'avons trouvé aucune solution. Cependant, nous pensons avoir compris d'où venait l'erreur. En effet, après avoir ajouté un utilisateur et demandé les informations du prochain individu, le *Scanner* écrasera les données de l'ancien utilisateur afin de stocker les nouvelles données, d'où l'affichage de la fiche récapitulative du dernier utilisateur uniquement.

Le deuxième mode du menu interactif est de demander à l'utilisateur le nom du fichier texte à traiter. En effet, notre programme peut calculer l'empreinte carbone à partir d'un objet *ObjectOutputStream* qui traduit l'objet sérialisable en un flux d'octets. On demande à l'utilisateur d'entrer le nom du fichier texte à traiter. Bien sûr, ce dernier doit être dans le même dossier que le projet. Nous avons décidé de créer un fichier texte assez simple qui est constitué de différentes parties : Nom, Prénom, Logement, Transport, Bienconsommation, Services Publics, Alimentation.

Afin de récupérer les données dans le fichier, nous appelons le deuxième constructeur de la classe *Utilisateur*qui prend en argument un *String* qui représente le nom du fichier. Le principe du constructeur est assez simple. Il initialise un *BufferReader* qui permettra de lire ligne par ligne le texte grâce à une boucle **While()**. La boucle est aussi composé d'un bloc **try{}** et **catch{}** afin de gérer les exceptions au cas où le fichier (*FileNotFoundException*) est introuvable et le cas où la méthode **read()** lève une *IOException*. Le code teste si un mot clé spécifique est présent dans la ligne afin de récupérer l'information importante, cela est réalisé grâce à la méthode de la classe *String* **contains().** Si le mot-clé apparaît dans la ligne, on sépare alors la ligne en deux parties grâce à la méthode split(), qui divise la ligne en fonction du caractère « : », car c'est après ce dernier que la donnée est située.

Ainsi, plusieurs blocs if{} sont utilisés afin de récupérer toutes les informations du fichier texte, afin de créer l'utilisateur

Exemple :

```
//récupère la superficie et la classe énergetique s'il a un logement
if (line.contains(s: "Superficie") || line.contains(s: "Classe énergétique")){
    if (line.contains(s: "Superficie")){
        String [] elemline = line.split(regex: ":");
        superficie = Integer.parseInt(elemline[1]);
    }
    if(line.contains (s: "Classe énergétique")){
        String [] elemline = line.split(regex: ":");
        ce = CE.valueOf(elemline[1]);
        Utilisateur.logements.add(new Logement(superficie,ce));
    }
}
```

Population :

Nous avons décidé de créer une classe population afin de gérer le calcul de plusieurs utilisateurs. En effet, cette classe prend comme attribut une *Collection* d'Utilisateurs qui regroupera tous les utilisateurs en une liste. Cette classe regroupe, les impacts énergétiques totaux de chaque classe et aussi l'impact énergétique moyen. Le constructeur de la classe effectue tous les calculs concernant les impacts. Il parcourt la liste d'utilisateurs et récupère l'impact de chaque classe.

La classe Population possède deux méthodes. La première est **DetaillePopulation()**. Celle-ci parcourt la liste d'utilisateurs et donne une fiche récapitulative de chaque utilisateur en affichant son Nom, Prénom et l'impact énergétique de chaque classe. La seconde est **DecisionMairie().** Cette méthode affiche une décision concernant la population entière et son impact énergétique. Elle met en place de nouvelles taxes, conseils… Si l'impact énergétique d'une des classes est beaucoup trop élevé. Nous avons pris comme référence la moyenne des impacts énergétiques des Français provenant du document fourni dans l'énoncé.

Interface Graphique :

Avant de faire la population, nous avions essayé d'implémenter l'interface graphique sans résultat concluant. Néanmoins, après plusieurs recherches, nous avons fini par s'approprier les methodes permettant la création d'une interface graphique via l'utilisation des classes JFrame et JDialog notamment. L'outil qui nous a le plus aider dans la création de cette interface est le site https://zestedesavoir.com/tutoriels/646/apprenez-a-programmer-en-java/558_java-et-la-programmation-evenementielle/2712_les-menus-et-boites-de-dialogue/#1-8665_les-boites-de-dialogue sur lequel il était possible d'avoir accès a un bon nombre d'exemples qui permettaient une compréhension plus rapide de chacune des notions.

L'interface graphique se base sur une classe *Accueil* qui permettra l'affichage des premières fenêtres/boites de dialogue d'introduction mais également toutes les fenêtres qui permettront la récupération des données de l'utilisateur et l'affichage du résumé de ses consommations avec l'impact associé à chacun des postes de consommation ainsi que quelques conseils dans le cas où son empreinte carbone excède l'empreinte carbone moyenne des français .

Le premier élément qui s'affiche sur l'écran de l'utilisateur n'est qu'a titre informatif et est suivi par un formulaire demandant l'identité de l'utilisateur. Nous avions essayé d'implémenter ce formulaire à l'aide d'une JFrame. Nous n'avions pas reussi a trouver un moyen d'afficher les 2 panels *panNom* est *panPrenom* dans la frame. Nous avons donc fini par utiliser la classe JDialog qui permettait un agencement des panels de manière plus intuitive et qui permettait surtout d'afficher l'ensemble des éléments dont nous avions besoin. Les donnée sont alors stockées dans les JTextFields *nom* et *prenom* et on pourra les récupérer plus tard pour la création de l'utilisateur grâce à la méthode **getText()**.

Lorsque l'utilisateur appuie sur le bouton suivant, la fenêtre sur laquelle on était est supprimée et une nouvelle fenêtre *Présentation* est créée et s'affiche sur l'écran de l'utilisateur. Cette fenêtre de présentation présente 5 boutons, chacun représentant un poste de consommation, et est évolutive. En effet lorsqu'un utilisateur appuie sur un bouton, une fenêtre (JDialog ou JOptionPane) prend la place de notre fenêtre présentation et affiche des questions liées au poste de consommation sélectionné. Les données entrées par l'utilisateur sont testées grâce à des **try{}** et des **catch{}** eux meme inclus dans un **while()**. Lorsque toutes les valeurs entrées respectent les exceptions définies

précédemment, le bouton sur lequel l'utilisateur a appuyé disparait de la fenêtre de présentation (puisque les données ont déjà été recueilli), cette fenêtre réapparait et l'utilisateur peut passer à une autre classe de consommation.

Comme mentionné plus haut, l'activation d'un bouton permet d'ouvrir le formulaire du poste de consommation qui lui est associé. Tous ces formulaires se ressemblent en terme de création (utilisation exclusive des JOptionPanel) sauf pour les postes de consommation de *Transport* et *Logement*. Dans ces 2 cas, l'utilisateurs peut avoir plusieurs éléments de ces classes il fallait donc trouver un moyen judicieux pour pouvoir récupérer les informations de chacun d'entre eux sans que la récupération de données soit trop encombrante. C'est pour cela qu'on a décidé d'utiliser des Jdialog sur lesquels on récupèrera l'ensemble des données nécessaires a la création d'un Logement ou d'un Transport. Ce formulaire s'affichera autant de fois que l'utilisateur possède de voiture ou de logement avec pour chacune de ces pages un nouveau titre (pour le $i^{ème}$ logement par exemple, on aura « logement n°i).

Lorsque tous les postes de consommation ont été traités et que tous les boutons ont été retirés de la fenêtre de présentation, cette fenêtre est définitivement supprimées et on passe a la derniere étape de l'interface graphique, l'affichage du bilan de la consommation de l'utilisateur et des potentiels conseil qu'on peut lui apporter. On peut passer à cette étape à partir de n'importe quel poste de consommation traité en dernier grâce a la derniere condition présente dans chacune de ces « pages » qui demande a passer a la page finale lorsque la fenêtre de présentation ne possède plus aucun bouton.

Deux dernières pages vont s'afficher, la JFrame de *Fin de programme* et la JDialog qui apparait lorsque l'utilisateur appuie sur le bouton « Valider » de la JFrame mentionnée et qui affichera le résumé et les conseil. Peu importe sur quel bouton l'utilisateur appuiera, la JFrame Fin de Programme sera supprimée définitivement et le programme sera terminé.

Un détail à rajouter, l'ensemble des JFrame qui ont été créée dans cette partie possède un menu dans lequel on peut trouver un item permettant de quitter le calculateur d'empreinte carbone. On peut également quitter le calculateur grâce au raccourci Ctrl+Q.