JAVA CPOO 2A 2021/2022

Commentaires sur la conception du projet de Simulation

Le digramme UML du projet est disponible dans le dossier source du projet, et porte le nom *UMLProject.png*.
Aussi, d'autres commentaires plus généraux sur ce projet sont disponibles dans le *README.md*. Enfin, le projet est disponible sur mon GitHub, au lien suivant: https://github.com/atxr/CPOO2A
Note: le repo github sera rendu public à partir de la deadline du projet, ie le 8 novembre.

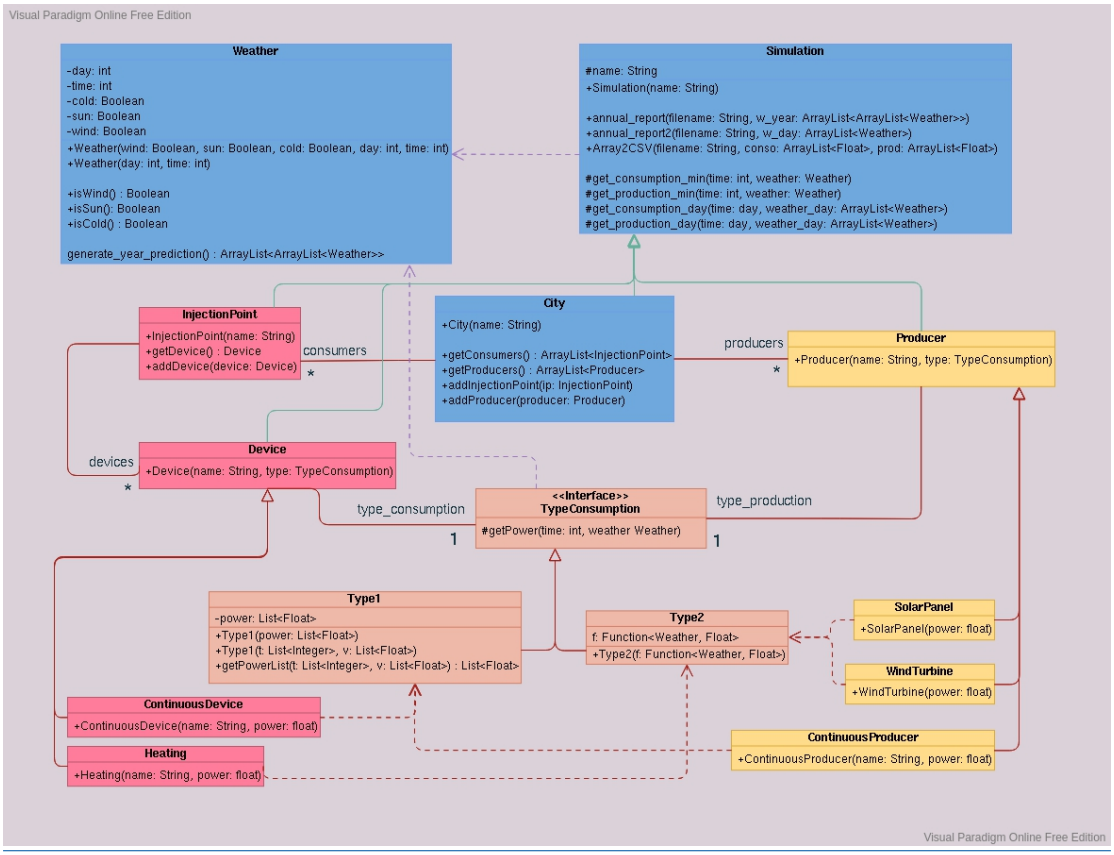

Diagramme UML du projet de Simulation de production/consommation d'éléctrécité.

Ce projet a été divisé en 4 packages principaux, qui sont:
- Simulation
- Devices
- Producers
- Type

Package Simulation

Le package Simulation, représenté en bleu sur le diagramme représente toutes les classes principales de la simulation de la ville. La classe abstraite Simulation est une classe permettant de décrire le fonctionnement de base de tous les objets par la suite qui pourront être simulés, et qui pourront donc générer un rapport journalier ou annuel, tel que City, InjectionPoint, Device ou encore Producer.
Le choix de classe abstraite m'a permit de coder directement dans la classe Simulation les

fonctions générant les rapports au format CSV de manière assez générale, tout en laissant les fonctions calculant la puissance émise des appareils aux classes filles.

Cette classe abstraite à une dépendance en la classe Weather, qui représente elle tous les facteurs extérieurs pouvant influer sur la simulation à un temps donnée, tel que la température, le vent ou encore la présence de soleil.

Enfin, City est une classe regroupant simplement un tableau de points d'injections et un autre de producteurs d'électricité.

Package Devices & Package Producers

Les packages devices et producers (rouge et jaune sur le digramme UML), bien que séparés, sont finalement assez similaire sur le point de vue de l'implémentation. La différence principale est l'existence d'une classe supplémentaire pour les Devices, qui est InjectionPoint, représentant une liste de Device. Cette classe me permet de générer un rapport pour une seule maison par exemple.

En effet, les classes InjectionPoint, Device et Producer héritent de la classe abstraite Simulation. Il est donc possible de générer des rapports, et elles implémentent les fonction manquantes permettant de calculer le consommation/production sur une durée donnée. Aussi, j'ai implémenté des classes toutes faites représentant des producteurs ou des appareil consommateurs d'électricité pour avoir une meilleure clarté de code lors de la construction de la ville.

Enfin, j'aimerais revenir sur mon choix de séparer la classe Producer et Device malgré leur similarité. En effet, le code semble au premier abord assez redondant, et on aurait pu imaginer une seule classe avec des puissances positives pour les producteurs et négatives pour les consommateurs.J'ai fait ce choix premièrement par clarté d'un point vu du problème et pas du code. Il me semblait plus judicieux de séparer ces deux instances pour la compréhension de l'utilisateur. Mais j'ai aussi fait ce choix par modularité, dans l'optique d'une potentielle extension du code qui pourrait faire diverger ces deux classes, comme par exemple l'ajout de "fuite" entre le producteur et le consommateur.

Package Type

Le dernier package, Type (en orange sur le diagramme UML), représente le type de consommation d'un consommateur/producteur. L'interface TypeConsumption et ses classes héritées Type1 et Type2 permettent de donner un modèle de calcul pour la puissance instantané d'une appareil/producteur.

Le Type1 représente un consommation/production périodique, tandis que le Type2 représente une consommation/production dépendante de paramètre extérieurs, symbolisés ici par la classe Weather. Cette dépendance est représentée dans le Type2 par une fonction, prenant en paramètre les conditions météos à un instant donné et calculant la puissance instantanée ainsi générée.

Le choix a été fait ici de symbolisé les paramètres météo par des Boolean, et non pas des Float, car j'ai pensé que la contribution de fonctions plus complexe prenant des nombres en paramètres n'apporterait que beaucoup plus de lignes de code pour un gain en précision assez négligeable. On pourrait donc imaginer comme extension de mon simulateur une prise en compte d'un modèle plus complexe en abandonnant cette description avec des Boolean.