Nous choisissons un design modulaire où la cohésion dans la classe est forte, les relations entre les classes sont faibles et les séparations sont claires visuellement afin d’avoir un programme plus compréhensible. Nous implémentons des classes abstraites car nous avons des modules ayant des similarités, comme les résidents, les compagnies privées et les agents publics, qui ont des éléments en commun, par exemple, le nom, l’adresse, le courriel, le numéro de téléphone et quelques autres. Des classes d’instanciations comme Requetes et Travaux utilise l’encapsulation, afin de créer de nouvelles instances où les informations ne peuvent pas être modifiées sans utiliser les setters données par la classe, et les informations ne peuvent pas être vus sans utiliser les getters. Avec toutes ces fonctionnalités que nous utilisons, une conception qui permet une plus grande docilité dans les changements futurs sur les classes, les méthodes et les variables est donc plus facile à gérer que d’autres modèles qui restreint les modifications possible sur le système. Les répercussions sur des nouveautés apportées au programme sont alors moins négatives et plus simple à régler en suivant ce design. Si de plus gros changements ont besoin d’être faits, par exemple rajouter des villes autres que Montréal dans notre système, rajouter d’autres types d’utilisateurs, modifier les envois des requêtes et des travaux pour qu’ils soient aussi envoyés par lettre, alors notre choix initial permet de faire ces modifications avec une plus grande facilité.