**Discussion et justification de la qualité du design**

Dans le design du devoir 2, on voit que les classes semblent être regroupées principalement par fonctionnalité et une compréhension basique des besoins du système. Le diagramme du 2e devoir tend à avoir des classes avec un peu trop de responsabilité, ainsi l’application de certains patrons de conception dans le design définit mieux les fonctionnalités des classes. Ainsi, la plus grosse différence entre les 2 designs est l’introduction des patrons de conception dans le 3e devoir. Les patrons de conceptions permettent d’organiser les classes par responsabilité claire et définie, ce qui améliore la séparation des préoccupations.

Cette conception introduit une meilleure modularité, avec des packages plus cohérents et des classes qui dépendent moins les unes des autres. Aussi, la décomposition des classes en des classes plus petites et approprié, cela rend le design plus gérable et extensible. On se dirige donc vers un couplage plus faible et une cohésion plus forte. Voici comment les différents patrons de conceptions améliorent notre design et les choix qu’on a fait pour les appliquer dans notre design initial :

Les fabriques centralisent la création d'objets, ce qui rend l'ajout de nouveaux types d'objets moins compliqué et évite de modifier le code existant. Cela allège la charge de travail lorsqu'on doit étendre notre système notamment si on doit ajouter infrastructures de voyage plus tard (qui risque d’arriver souvent grâce à l’évolution des villes).

Le patron d'état est spécialement utile pour gérer les différentes phases d'une réservation (siège/cabine). Chaque phase a un comportement bien défini, et le système peut passer d'une phase à une autre sans problème, ce qui nous donne une grande flexibilité dans la gestion du flux des réservations. Le visiteur permet d’exécuter des opérations qui peuvent varier selon l'objet sans avoir à changer les classes sur lesquelles elles opèrent. Cela signifie qu'on peut introduire de nouvelles actions sans modifier notre structure de classes, ce qui est un atout majeur pour la maintenance.

Le patron de l’observateur garantit la cohésion entre la partie client et la partie administratif. Pour les interactions utilisateur, le patron de commandes nous permet de traiter chaque action, comme une réservation ou une annulation, comme un objet à part entière. Ce faisant, on peut facilement annuler ou répéter des actions, ce qui apporte une robustesse et une sûreté dans la gestion des actions des utilisateurs. Le singleton nous aide à gérer les ressources uniques, comme la connexion à la base de données, en s'assurant qu'une seule instance gère toutes les opérations, optimisant ainsi les performances et la gestion des ressources.

Ainsi, l’intégration de ces patrons nous a permis d’optimiser la structure et la performance de notre système. Par rapport aux précédents designs, on a essayé de respecter au maximum le SRP en optimisant la fonctionnalité de chaque classe et en partageant les fonctions. Le principe ouverture/ fermeture et le LSP en utilisant les abstractions et les commandes. On a suivi aussi le ISP en créant des interfaces bien définies et spécifiques pour chaque fonctionnalité, évitant ainsi que les classes implémentant ces interfaces ne soient forcées de dépendre de méthodes dont elles n'ont pas besoin.

Bien qu’on a essayé d’appliqué ces principes de conception, notre design est loin d’être parfait. Malgré nos efforts pour réduire le couplage, certaines parties du système pourraient encore être trop interdépendantes, surtout si les interfaces ne sont pas correctement définies ou si les classes ne respectent pas complètement le principe de responsabilité unique. Ainsi, cela pourrait nuire à la maintenance et la réutilisabilité de certaines parties du design et du code.