RAKOTOARISOA Tahiriniaina Andrian
MBILIA Maurice
M1 Informatique IL
Année universitaire : 2020-2021

Rapport de projet ACO : CarTaylor

Choix d'implémentations V1 :

Category :

Pour CategoryImpl : l'implémentation de l'interface Category, nous avons utilisé seulement categoryName comme attribut qui représente le nom de cette catégorie.

PartType :

Pour PartTypeImpl, les attributs sont partName qui représente le nom du PartType et Category qui sera le categorie du PartType. De plus, nous avons rajouté la méthode printDescription qui sera utilisée pour la génération de la page HTML pour la description de la configuration.
Nous avons décidé de garder chaque catégorie pour chaque partType parce que c'est plus facile ainsi pour nous de savoir dans quelle catégorie le PartType y est associé.

Configuration :

Pour Configuration, elle contient selectPart qui représente les "part" sélectionné, categories qui représente toute les catégorie possible et un CompatibilityManager compatibilityMg. CompatibilityManager est utilisé pour savoir si la configuration est valide et/ou est complète, et donc un attribut privé pour ce dernier. Nous avons rajouté la méthode printDescription dans configuration pour pouvoir générer la page HTML de sa description.

CompatibilityManager :

Pour le rajout d'incompatibilité, nous avons décidé que les deux parts incompatibles soient incompatibles mutuellement l'une à l'autre. Accordé par le tableau dans le sujet du projet, nous ne pouvons pas ajouter une

incompatibilité ou un “requirement” à des “part“ qui sont dans les mêmes catégories (et qui est aussi plus logique).
CompatibilityManager a besoin de toutes les part pour pouvoir être instancié, et qui seront après utilisé pour l’initialisation dans les maps.
CompatibilityManagerImpl, pour addRequirements et addIncompatibilities, nous avons utilisé des map pour pouvoir faciliter l’association de la référence et de la liste de part cible. Nous avons donc décidé de créer deux maps : l’un qui est requirements qui représente les part requises et l’autre incompatibilities qui représente les incompatibilités pour les part.
Et vue que CompatibilityManager hérite de CompatibilityChecker il n’est pas nécessaire d’implémenter CompatibilityChecker comme toutes ses opérations seront implémentées dans CompatibilityManagerImpl.

Configurator :

Pour ConfiguratorImpl, nous avons utilisé des attributs privés pour les listes de catégories, part, configuration et pour compatibilityManager afin de pouvoir les garder en mémoire pour faciliter l’implémentation des opérations dedans.
De plus, pour l’instanciation de configurator, nous faison l’initialisation des catégories, des part, de configuration et de compatibilityManger.

Implémentation pour V2 :

Pour la V2, nous n’avons pas eu le temps de bien implémenter les choses, du coup nous n’avons ajouté que les interfaces et classes recommandées et apporté quelques modifications dans certaines classes d’implémentation.

Rapport des testes et user stories :

User story 1 : Dans le test ConfiguratorTest l’utilisateur peut accéder à la liste des catégories grâce à l’opération getCatégories().

Dans notre test, nous avons comparé deux “set” de catégorie avec le “set” que getCatégories() renvoie. L’un est censé passé (setc1) et l’autre échoué.

User story 2 : Dans le test ConfiguratorTest, l’utilisateur peut acceder à la liste des variants à partir d’une catégorie grâce à l’opération getVariants(Category category).
Dans le test, nous avons créé un “set” pour les variantes qui contient toutes les part de catégorie 1 : Engine et nous l’avons comparé avec la valeur de retour de l’opération getVariants(category1). Le premier teste passe mais le second est censé passé.

User story3 : Dans ConfigurationTest, l’utilisateur peut savoir si sa configuration est complete et/ou valide grâce à l’opération isComplete() et respectivement isValid().
Dans le test, nous avons fait deux cas pour les deux méthodes : l’un qui échoue et l’autre qui est censé passé mais qui échoue.

User story 4 : Dans ConfigurationTest l’utilisateur peut supprimer des part dans la configuration courante grâce à l’opération unselectPartType(Category categoryToClear)
Dans le test, nous avons ajouté le part11 : EG100, category : Engine, part21 : ED180, category : Engine et part22 : TA5 category : Transmission à la configuration. Quand nous avons appelé la méthode unselectPartType(category1), le seul part qui est présent dans la configuration est part22 vue qu’il a supprimé toutes les part de category1 : Engine.
Résultat du test : succès.

User story 5 : Dans CompatibilityManagerTest, l’utilisateur peut éditer la liste des part incompatibles/required dans l’application grâce aux opérations fournies dans CompatibilityManagerImpl.
Dans le test de addrequirementTest & addincompatibilities nous nous sommes assuré qu’on ne peut pas l’ajout de incompatibilité/requirement qui ont la même catégorie.

Résultat du test : succès.

User story 6 : Nous avons rajouté une opération dans l'interface de configuration et PartType : printDescription (printStream)
Dans configurationTest, un fichier file.htlm est généré cependant comme notre test isValid() et isComplete() échouent la page ne contient rien.
Résultat du test : échec.

Bilan des testes :

Teste succès :

CategoryTest, PartTypeTest : testGetName, testGetCategory, CompatibilityManagerTest, ConfigurationTest : testSelectedPart, unselectedPartTest.

Teste échoué : ConfiguratorTest, PartTypeTest : testEqualityOfPartTypes et les testes mentionnés tout en haut récemment.

Problème rencontré :

Nous voulons rajouter quelques fonctionnalités dans configurationImpl mais cette fonctionnalité était inaccessible lors de l'appel de cette méthode dans une autre classe.
La méthode .equals() semble ne pas fonctionner comme prévue. Ca aurait était mieux si on pouvait rajouter une operation equals dans certains interfaces.