COMPTE-RENDU TP
CAR-TAYLOR

Encadrant
CHARLES QUEGUINER

Auteurs

ATTOUBE YANN

DEGNI JOCELIN

M1 MIAGE

GROUPE DE TP : B

I) Présentation et contexte du projet

Dans le cadre de l'unité d'enseignement ''Analyse et Conception Objet'' il nous ait demandé de créer une petite application dénommé '' CarTailor ''. CarTailor est un configurateur de voiture développé en java permettant à un utilisateur de sélectionner plusieurs parties d'un véhicule afin de construire un véhicule spécifique. Les parties pouvant être configurées sont :

- Le type du moteur
- Le type de la transmission
- Le style intérieur
- Le style extérieur

Plusieurs possibilités existent pour chaque catégorie, et chaque possibilité est soumise à des contraintes de compatibilités ou de dépendance avec d'autres possibilités.

L'application comprend deux versions :

- Une première version permettant à un utilisateur de :
  - Demander la liste des catégories et en choisir une
  - Sélectionner une pièce dans cette catégorie et l'ajouter à sa configuration
  - Savoir si la configuration est valide ou non(respecte toutes les contraintes de compatibilité ou d'incompatibilité)
  - Supprimer une pièce de ma configuration
  - Et en tant qu'administrateur, modifier l'ensemble des pièces requises ou incompatibles
- Une seconde version qui est une évolution de la première permettant à l'utilisateur de :
  - Demander la description de sa configuration en un fichier html si celle-ci est complète et valide
  - Demander le prix de la configuration en euros même si elle est incomplète(mais elle doit être valide)
  - Sélectionner la couleur de l'extérieur provenant d'un sous ensemble produit par l'application (le sous ensemble de couleur dépend de la pièce extérieur que l'utilisateur aura choisi)

Ce rapport présente quelques étapes de la conception de cette application notamment les diagrammes de classe des deux versions, quelques diagrammes de séquences, et une synthèse des cas de test décrivant une simulation des choix qu'un client pourras effectuer.

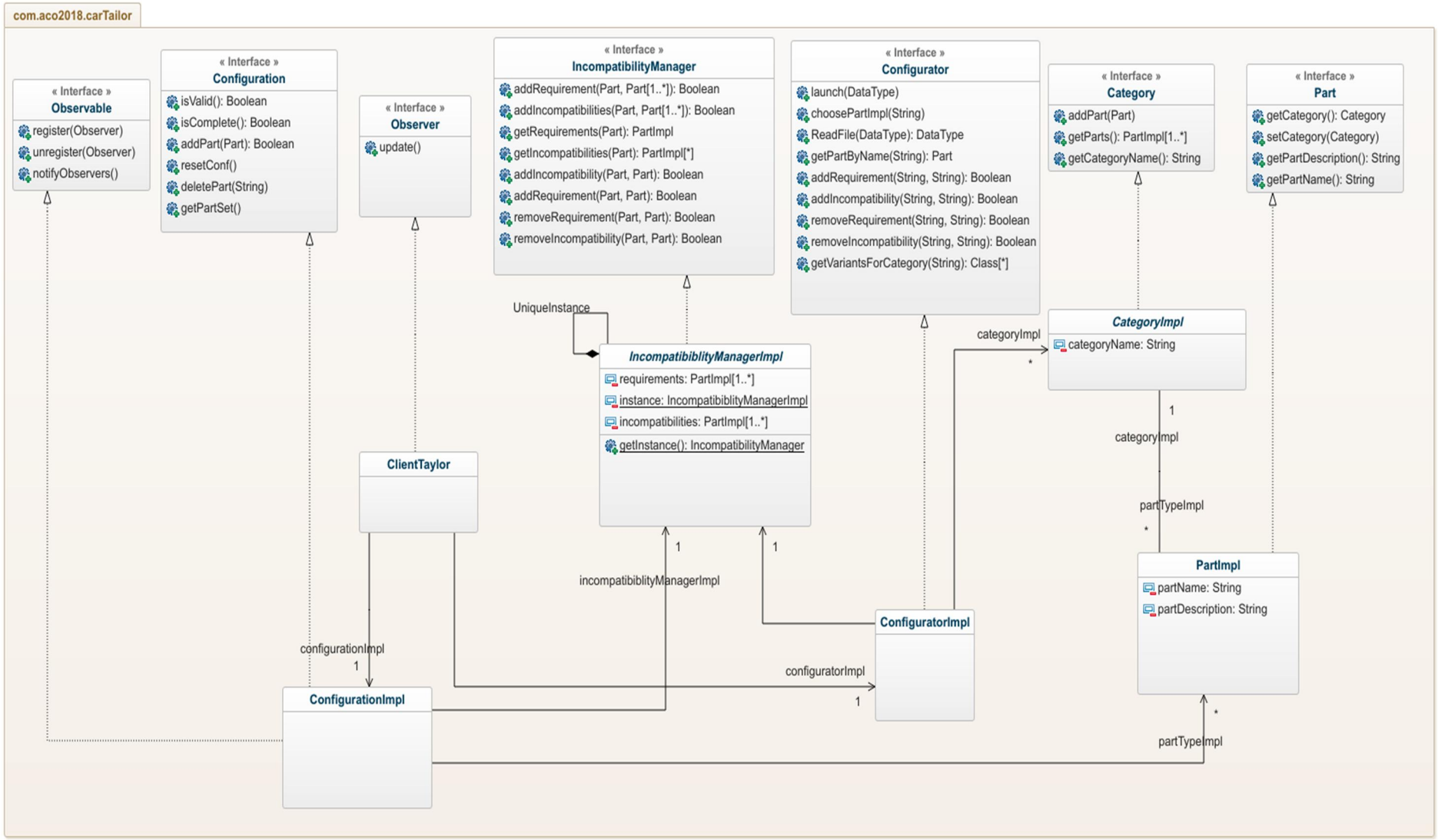
Pour le chargement du catalogue dans l'application, nous avons choisi de l'écrire dans un fichier JSON, l'application fait la lecture de ce fichier et crée le catalogue à partir de celui-ci.

Cela évite de créer à chaque fois à la ''main'' les catégories, et pour chaque catégories les différentes pièces.

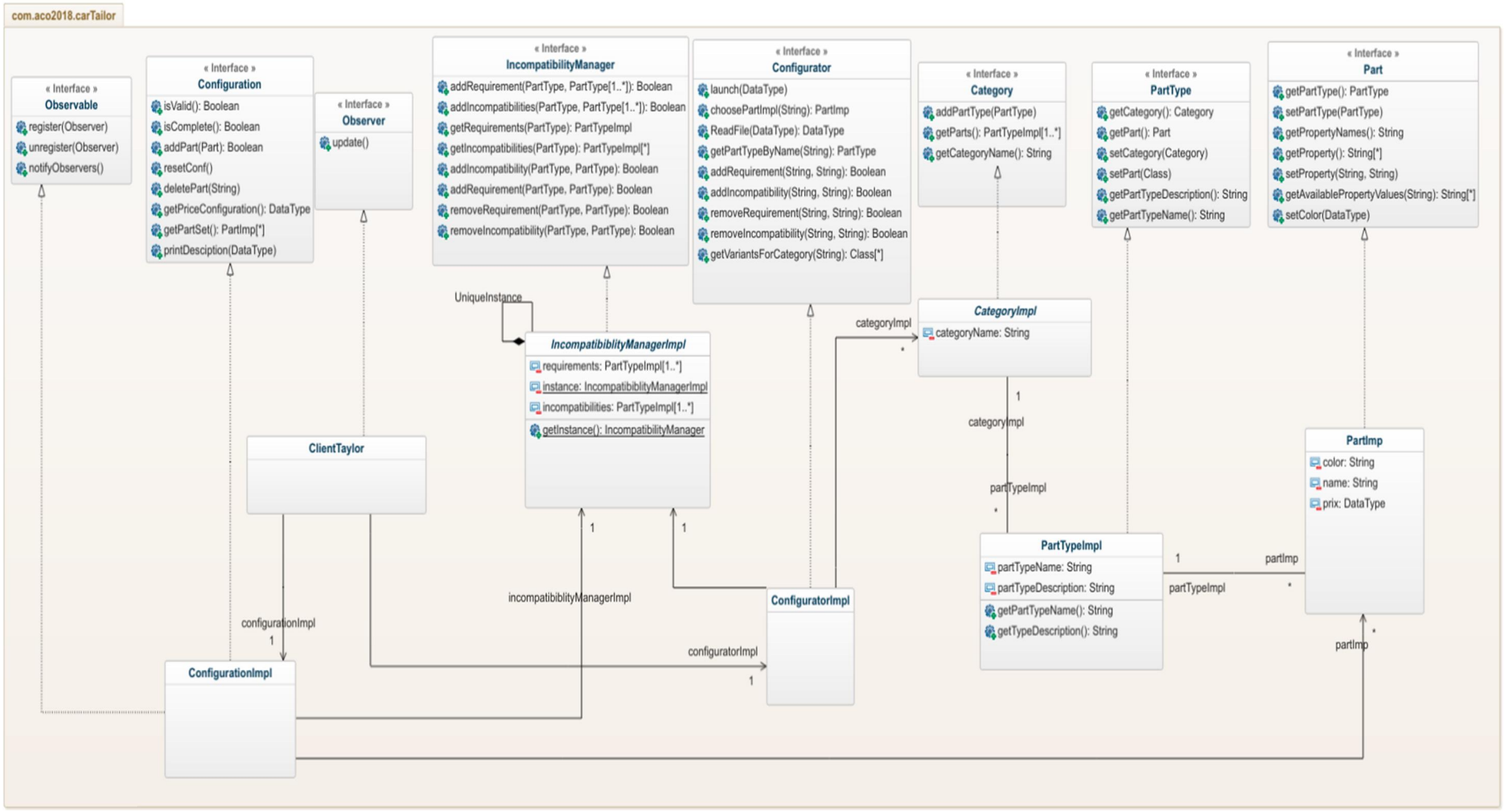
II) Conception UML

1) Diagrammes de classe

a) Version 1

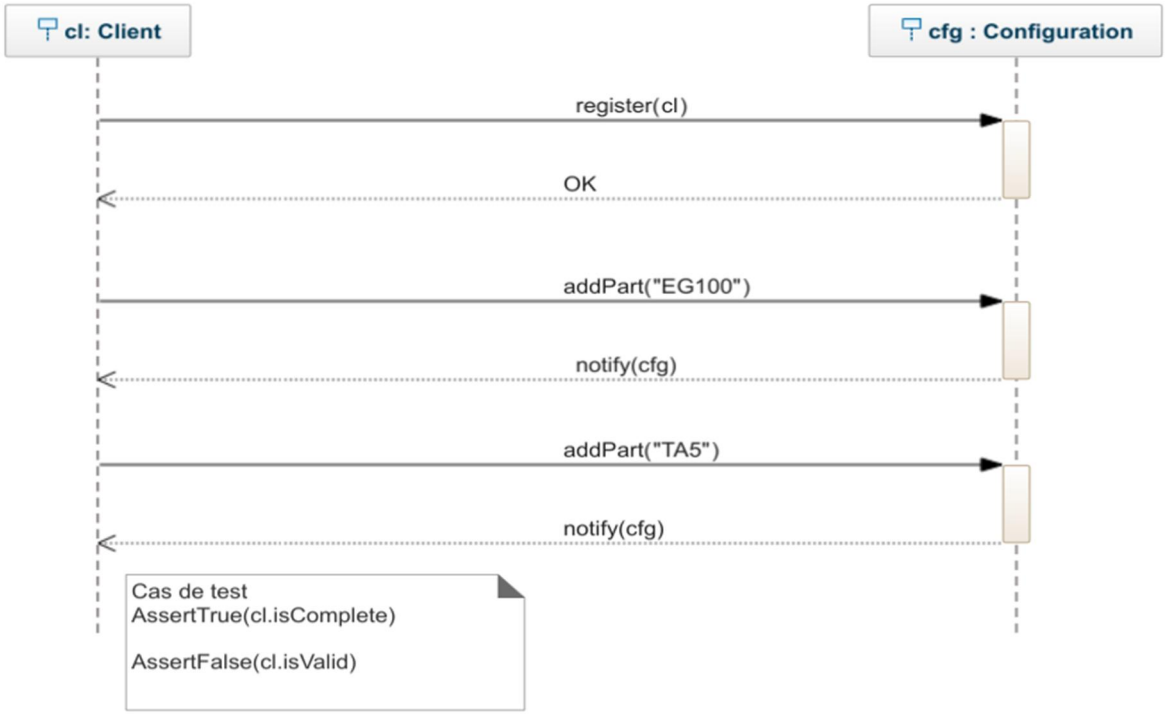

com.aco2018.carTailor
« Interface »
Observable
register(Observer)
unregister(Observer)
notifyObservers()
« Interface »
Configuration
isValid(): Boolean
isComplete(): Boolean
addPart(Part): Boolean
resetConf()
deletePart(String)
getPartSet()
« Interface »
Observer
update()
« Interface »
IncompatibilityManager
addRequirement(Part, Part[1..*]): Boolean
addIncompatibilities(Part, Part[1..*]): Boolean
getRequirements(Part): PartImpl
getIncompatibilities(Part): PartImpl[*]
addIncompatibility(Part, Part): Boolean
addRequirement(Part, Part): Boolean
removeRequirement(Part, Part): Boolean
removeIncompatibility(Part, Part): Boolean
« Interface »
Configurator
launch(DataType)
choosePartImpl(String)
ReadFile(DataType): DataType
getPartByName(String): Part
addRequirement(String, String): Boolean
addIncompatibility(String, String): Boolean
removeRequirement(String, String): Boolean
removeIncompatibility(String, String): Boolean
getVariantsForCategory(String): Class[*]
« Interface »
Category
addPart(Part)
getParts(): PartImpl[1..*]
getCategoryName(): String
« Interface »
Part
getCategory(): Category
setCategory(Category)
getPartDescription(): String
getPartName(): String
UniqueInstance
IncompatibliblityManagerImpl
requirements: PartImpl[1..*]
instance: IncompatibliblityManagerImpl
incompatibilities: PartImpl[1..*]
getInstance(): IncompatibilityManager
CategoryImpl
categoryName: String
categoryImpl
*
1
categoryImpl
partTypeImpl
*
PartImpl
partName: String
partDescription: String
ClientTaylor
1
1
incompatibliblityManagerImpl
configurationImpl
1
ConfigurationImpl
configuratorImpl
1
ConfiguratorImpl
*
partTypeImpl

b) Version 2

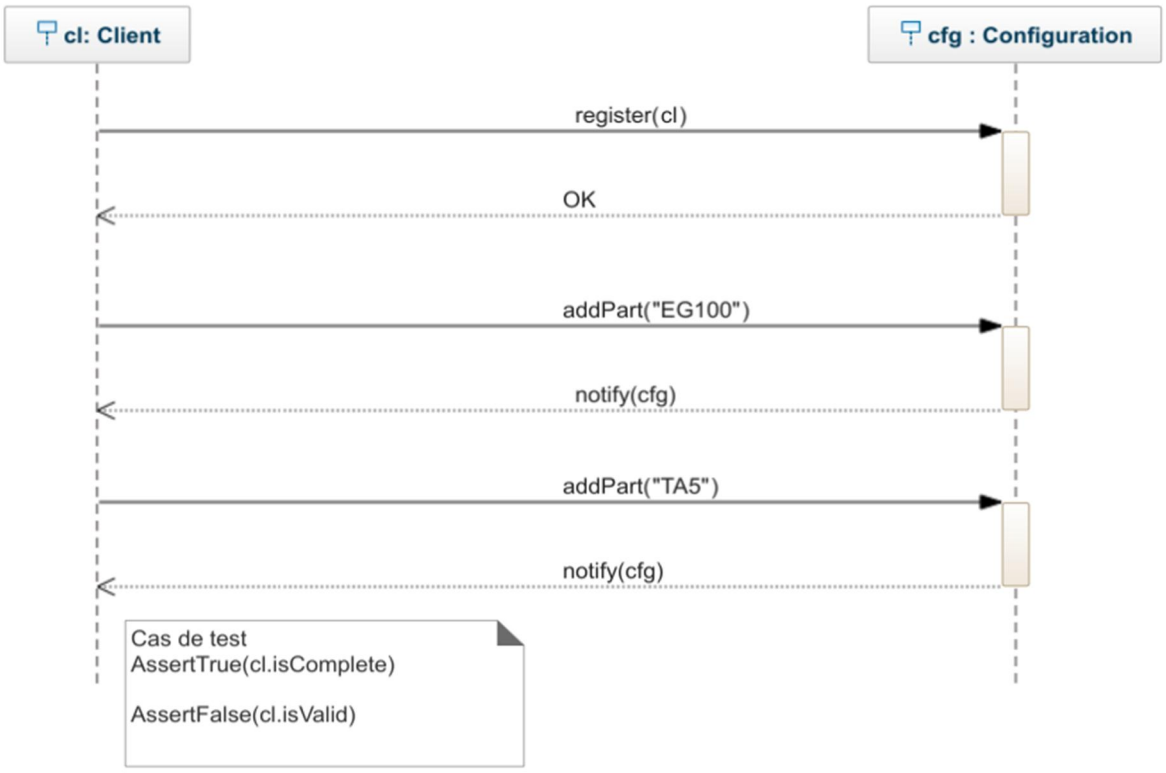

com.aco2018.carTailor
« Interface »
Observable
register(Observer)
unregister(Observer)
notifyObservers()
« Interface »
Configuration
isValid(): Boolean
isComplete(): Boolean
addPart(Part): Boolean
resetConf()
deletePart(String)
getPriceConfiguration(): DataType
getPartSet(): PartImp[*]
printDesciption(DataType)
« Interface »
Observer
update()
« Interface »
IncompatibilityManager
addRequirement(PartType, PartType[1..*]): Boolean
addIncompatibilities(PartType, PartType[1..*]): Boolean
getRequirements(PartType): PartTypeImpl
getIncompatibilities(PartType): PartTypeImpl[*]
addIncompatibility(PartType, PartType): Boolean
addRequirement(PartType, PartType): Boolean
removeRequirement(PartType, PartType): Boolean
removeIncompatibility(PartType, PartType): Boolean
« Interface »
Configurator
launch(DataType)
choosePartImpl(String): PartImp
ReadFile(DataType): DataType
getPartTypeByName(String): PartType
addRequirement(String, String): Boolean
addIncompatibility(String, String): Boolean
removeRequirement(String, String): Boolean
removeIncompatibility(String, String): Boolean
getVariantsForCategory(String): Class[*]
« Interface »
Category
addPartType(PartType)
getParts(): PartTypeImpl[1..*]
getCategoryName(): String
« Interface »
PartType
getCategory(): Category
getPart(): Part
setCategory(Category)
setPart(Class)
getPartTypeDescription(): String
getPartTypeName(): String
« Interface »
Part
getPartType(): PartType
setPartType(PartType)
getPropertyNames(): String
getProperty(): String[*]
setProperty(String, String)
getAvailablePropertyValues(String): String[*]
setColor(DataType)
UniqueInstance
IncompatibliblityManagerImpl
requirements: PartTypeImpl[1..*]
instance: IncompatibliblityManagerImpl
incompatibilities: PartTypeImpl[1..*]
getInstance(): IncompatibilityManager
categoryImpl
CategoryImpl
categoryName: String
1
categoryImpl
*
partTypeImpl
*
ClientTaylor
1
1
incompatibliblityManagerImpl
ConfiguratorImpl
configurationImpl
1
configuratorImpl
1
PartTypeImpl
partTypeName: String
partTypeDescription: String
getPartTypeName(): String
getTypeDescription(): String
1
partImp
partTypeImpl
*
PartImp
color: String
name: String
prix: DataType
partImp
*
ConfigurationImpl

2) Diagrammes de séquences

a) Enregistrer un observer sur la configuration



b) Tester la compatibilité entre EG100 et TM5

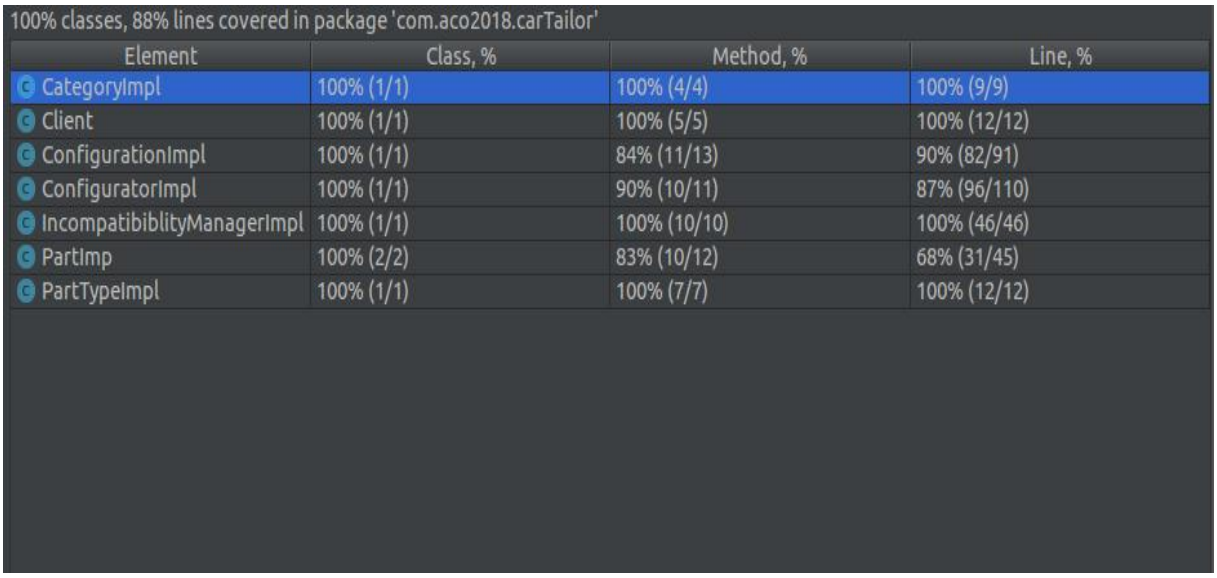


III) Synthèse des cas de tests

Notre application regroupe plusieurs tests tels que :

- Dest tests unitaires pour tester la classe Configuration(Tester les différentes méthodes qu'une configuration pourra effectuer)
- Des tests pour la classe Configurateur (Vérifier que le fichier JSON a bien été lu et que le catalogue est représenté en entier etc…)
- Et une grande classe de test qui simule les échanges qu'un client pourra avoir avec l'application en version 1 comme en version 2

Résultats de la couverture des tests



100% classes, 88% lines covered in package 'com.aco2018.carTailor'

Element	Class, %	Method, %	Line, %
CategoryImpl	100% (1/1)	100% (4/4)	100% (9/9)
Client	100% (1/1)	100% (5/5)	100% (12/12)
ConfigurationImpl	100% (1/1)	84% (11/13)	90% (82/91)
ConfiguratorImpl	100% (1/1)	90% (10/11)	87% (96/110)
IncompatibiblityManagerImpl	100% (1/1)	100% (10/10)	100% (46/46)
PartImp	100% (2/2)	83% (10/12)	68% (31/45)
PartTypeImpl	100% (1/1)	100% (7/7)	100% (12/12)