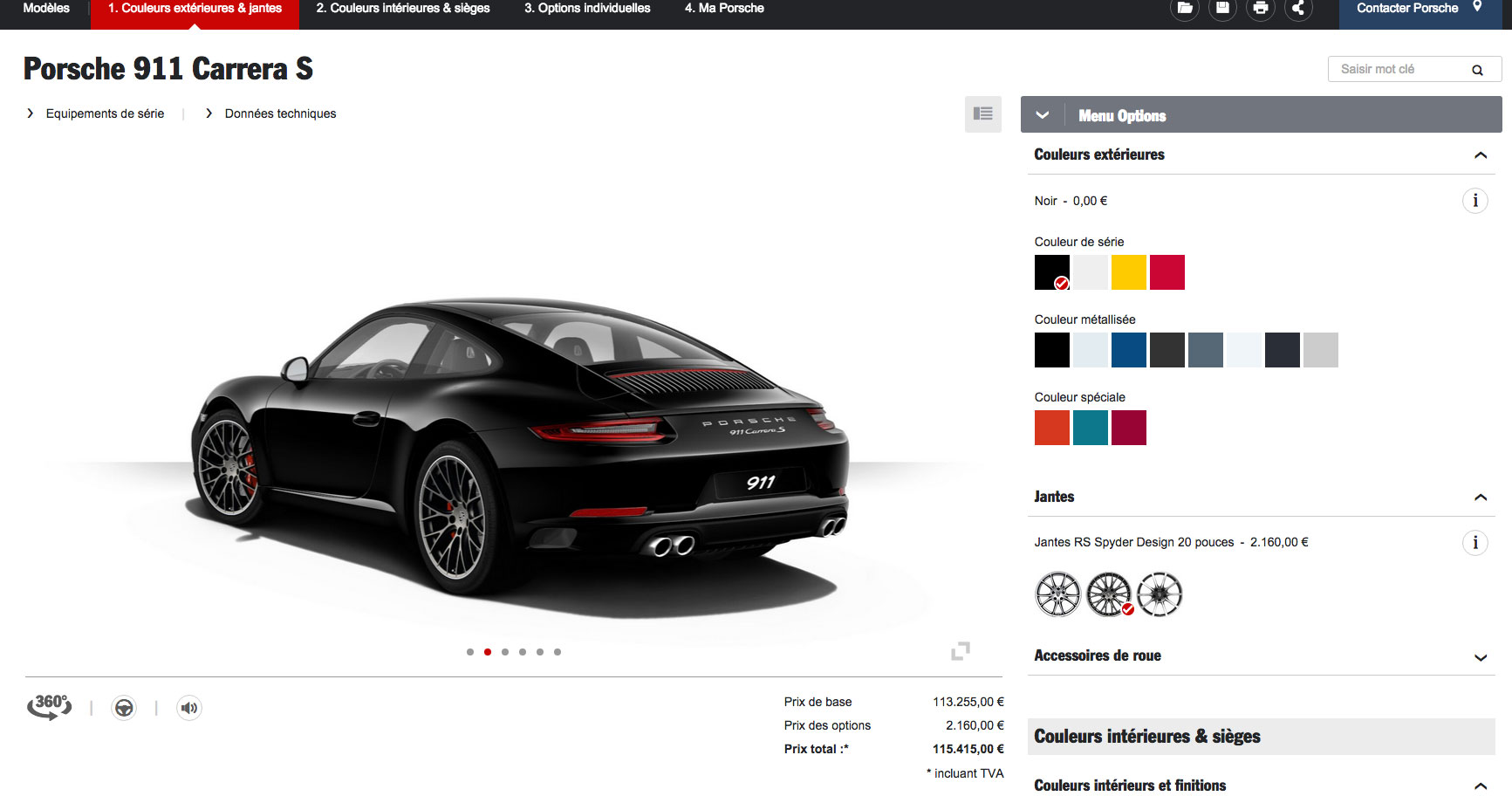
Car Tailor

Configurateur de voiture

ACO

SCRIMALI Gaëtan | MANDE Guillaume | LEGROUX Thibaut

M1 MIAGE Groupe 1B

# Sommaire

## Context

Dans le cadre de notre Master 1 MIAGE et du module d’ACO, nous avons réalisé l’implémentation d’un configurateur de voiture. Ce projet nous a demandé tout d’abord demandé la conception d’un diagramme de classe puis d’un diagramme de séquence. Nous nous sommes ensuite penchés vers la programmation du code.

## Modèle Uml

Le configurateur de voiture possède plusieurs caractéristiques. Tout d’abord, nous pouvons distinguer quatre catégories :

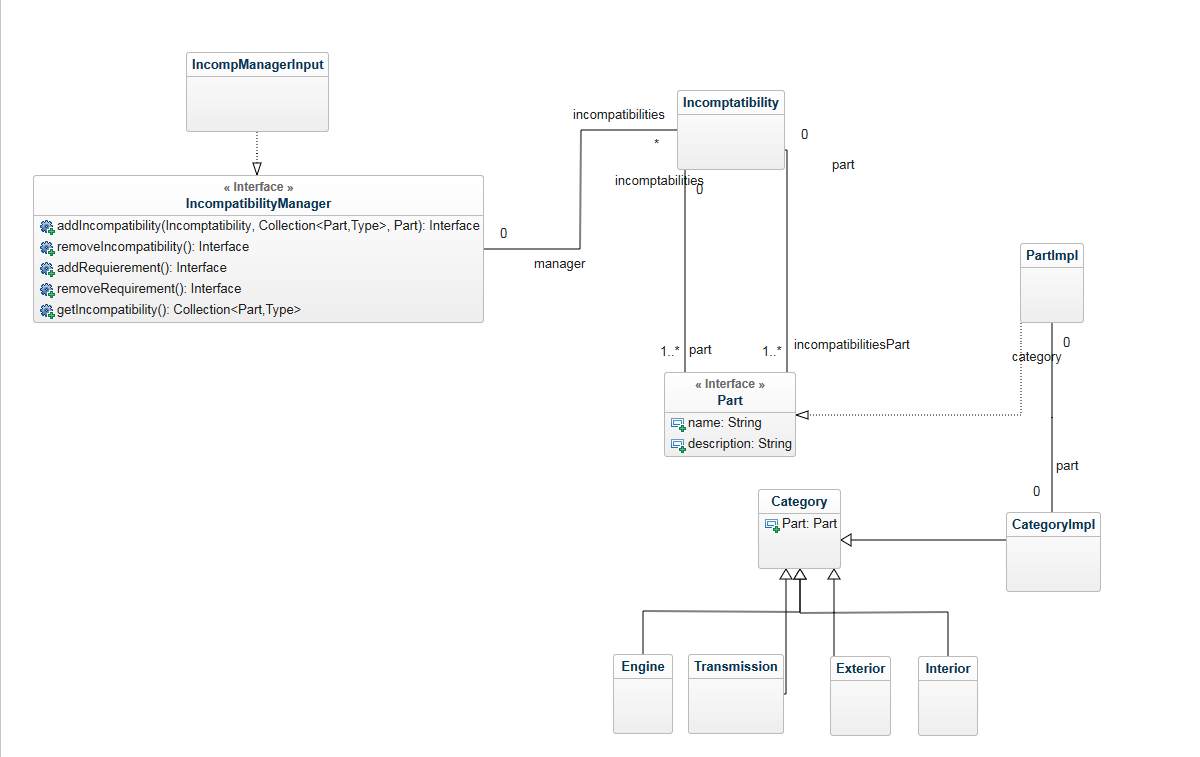
* Engine 🡪 Le moteur
* Transmission
* Exterior 🡪 La peinture extérieur de la voiture
* Interior 🡪 Le niveau de finition intérieur de la voiture.

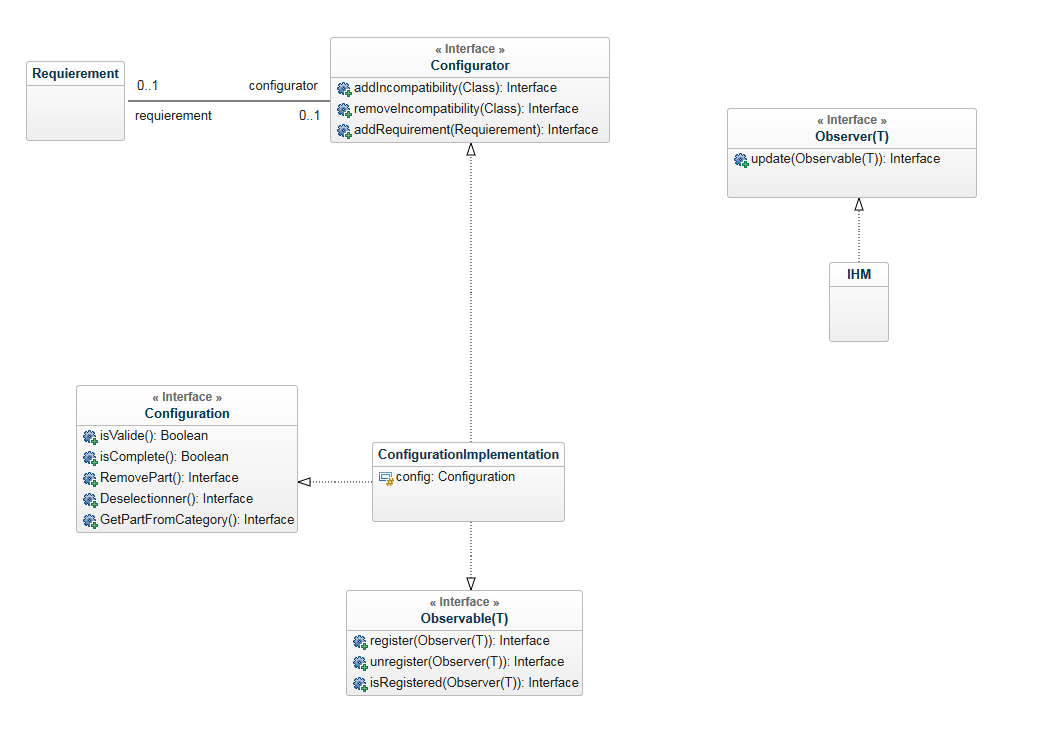
Apparaissent ensuite deux notions très importantes pour la conception et l’implémentation de cette configuration de voiture. En effet, les prérequis à la sélection d’une partie et les incompatibilités entre en ligne de compte dans la création du produit final.

Nous avons modélisé deux diagrammes UML qui correspondent avec les attentes formulées en TD.

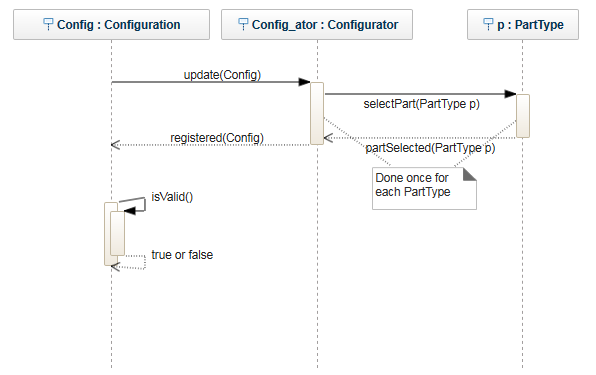
Le premier diagramme explicite la gestion des incompatibilités entre les différentes parties. Par exemple un moteur dit « sport » ne pourra pas être associé à un certain niveau de finition intérieur et à une certaine peinture qui ne correspondent pas à une finition « sport ». Par ailleurs, un moteur hybride ne pourra être associé à une transmission uniquement essence.

Ces différents cas sont donc des incompatibilités techniques.



Le second diagramme de classe explicite le fonctionnement des prérequis à la sélection d’une pièce, c’est-à-dire la nécessité de sélectionner une pièce en particulier pour avoir accès à la sélection d’une autre pièce, par exemple, pour avoir accès à une peinture extérieur sport, le client devra impérativement choisir les moteurs de la gamme sport. Il représente également la partie centrale du configurateur, c’est-à-dire la gestion de la sélection des différentes parties et donc la vérification des prérequis et des incompatibilités entre les différentes pièces. Ce diagramme de classe modélise donc le configurateur et la vérification de la validité d’une configuration.

De plus, un diagramme de séquence a été modélisé. Il explicite le fonctionnement global de l’application. C’est-à-dire le scénario nominal, quand tout ce passe bien et que l’utilisateur/potentiel client sélectionne une configuration valide.



## Explications des choix UML

Le premier diagramme de classes nous permet de comprendre que nous avons articulé notre projet autour de la classe IncompatibilityManager. En effet elle est essentielle aux autres. La gestion des incompatibilités est donc reliée à la gestion des différentes catégories via la classe Part (qui sera implémenté dans la V2). Nous avons procédé ainsi dans un souci d’accessibilités aux différentes catégories. La gestion des incompatibilités est donc réalisée dans cette partie du diagramme de classes et avant la phase de configuration. Ainsi, au moment de la programmation de la configuration de notre voiture, nous n’avons pas à nous occuper de cela.

Dans le second diagramme UML nous avons décidé de faire prendre à la classe ConfigurationImplementation (qui est la classe implémentant l’interface Configuration) une place centrale. En effet le client aura accès au système via cette classe (qui sera par la suite mit en lien avec l’IHM). Autre classe, autre utilité : Configurator, elle se charger de gérer deux choses. Tout d’abord, elle récupère les Incompatibilités mais aussi gérer l’autre facette importante du configurateur : les prérequis à la sélection d’un composant de la voiture.

Enfin et seulement après la gestion de tous ces aspects, la classe ConfigurationImplementation se chargera de récupérer la configuration du client et de vérifier si elle est valide. Donc si la sélection faite respect les prérequis, les incompatibilités et si elle est complète, c’est-à-dire, si elle comprend un élément de chaque catégorie.

Concernant le digramme de séquence, nous avons décidé de nous concentrer sur un fonctionnement nominal du configurateur de voiture. Ici, le client sélectionne des éléments compatible et où les prérequis sont respectés. Le client choisi alors les quatre éléments de ça futur voiture. Puis le configurateur se charge de vérifier que tous les différentes exigences sont respectées. Et donc si la configuration est valide ou non.

## Code

Le travail de programmation a quasiment été exclusivement tourné vers la V1 demandée. Toutefois quelques classes, comme Requirement, Incompatibility et IncompatibilityManager ont été programmées.

Nous nous concentrerons donc ici sur les choix fais pendant la réalisation de la V1.

La programmation de la V1 implique principalement trois classes : ConfiguratorImpl, ConfigurationImpl et CompatibilityManagerImpl. Ces trois classes représentent la base de la V1 et permettent le bon fonctionnement

La V1 doit contenir les implémentations de plusieurs classes :

- CategoryImpl  
- PartTypeImpl  
- ConfigurationImpl  
- ConfigurationImplTest  
- ConfiguratorImpl  
- ConfiguratorImplTest  
- CompatibilityManagerImpl (CompatibleManagerImpl pour nous)  
- CompatibleManagerImplTest

CategoryImpl permet la gestion des catégories mais aussi la gestion des différentes parties de chaque catégorie. C’est-à-dire les nombreuses références de chaque catégorie. Nous pouvons donc ajouter ou supprimer une référence mais aussi savoir si une référence existe. La classe comprend aussi des getters. Cela nous permet une plus grande flexibilité.

PartTypeImpl comprend une suite d’assesseurs afin de modifier l’appartenance d’une référence à une catégorie ou de récupérer la description et le nom d’une référence.

ConfigurationImpl, cette classe contient des méthodes d’ajout et de suppression d’une référence au catalogue. La classe de test associé est ConfigurationImplTest

ConfiguratorImpl cette classe contient le catalogue des références du configurateur. Nous avons aussi décidé d’ajouter des assesseurs pour accéder aux catégories et aux références contenu dans celles–ci. La classe de test associé est ConfiguratorImplTest

CompatibleManagerImpl comme dit plus tôt, cette classe fait la liaison entre toutes les parties du projet. Nous y retrouvons principalement les méthodes de vérification de configuration. La classe de test associé est CompatibleManagerImplTest.

A noté que les classes ayant le suffixe « Impl » sont des classes dépendant de l’interface correspondante.

Par ailleurs certaines classes et fonction ayant un rapport direct avec la V2 ont été implémentées. Nous faisons ici allusion aux fonctions de gestion des incompatibilités (Incompatibility) et de prérequis (Requirement). Ainsi la classe CompatibleManagerImpl comprend donc les méthodes de gestion de ces différentes exceptions nécessaire au bon fonctionnement du configurateur de voiture.

## Résultat

Couverture de cas de test 🡪 option test «with converage »

## La suite ?

Comme nous pouvons le remarquer à la lecture de ce présent rapport, nous nous sommes concentrés sur la bonne réalisation de la V1. Cela signifie que peu de fonction de la V2 ont été implémentées. Par exemple l’IHM qui doit à terme remplacer la classe ConfigurationImpl n’a pas été implémenté.

La réflexion et la programmation autour de la V2 de notre CarTailor consiste le principal axe de développement. Cela comprend un codage plus affiné des fonctions lié aux Incompatibilités, Requirement mais aussi à toute la facette Part du configurateur.