



Car Tailor

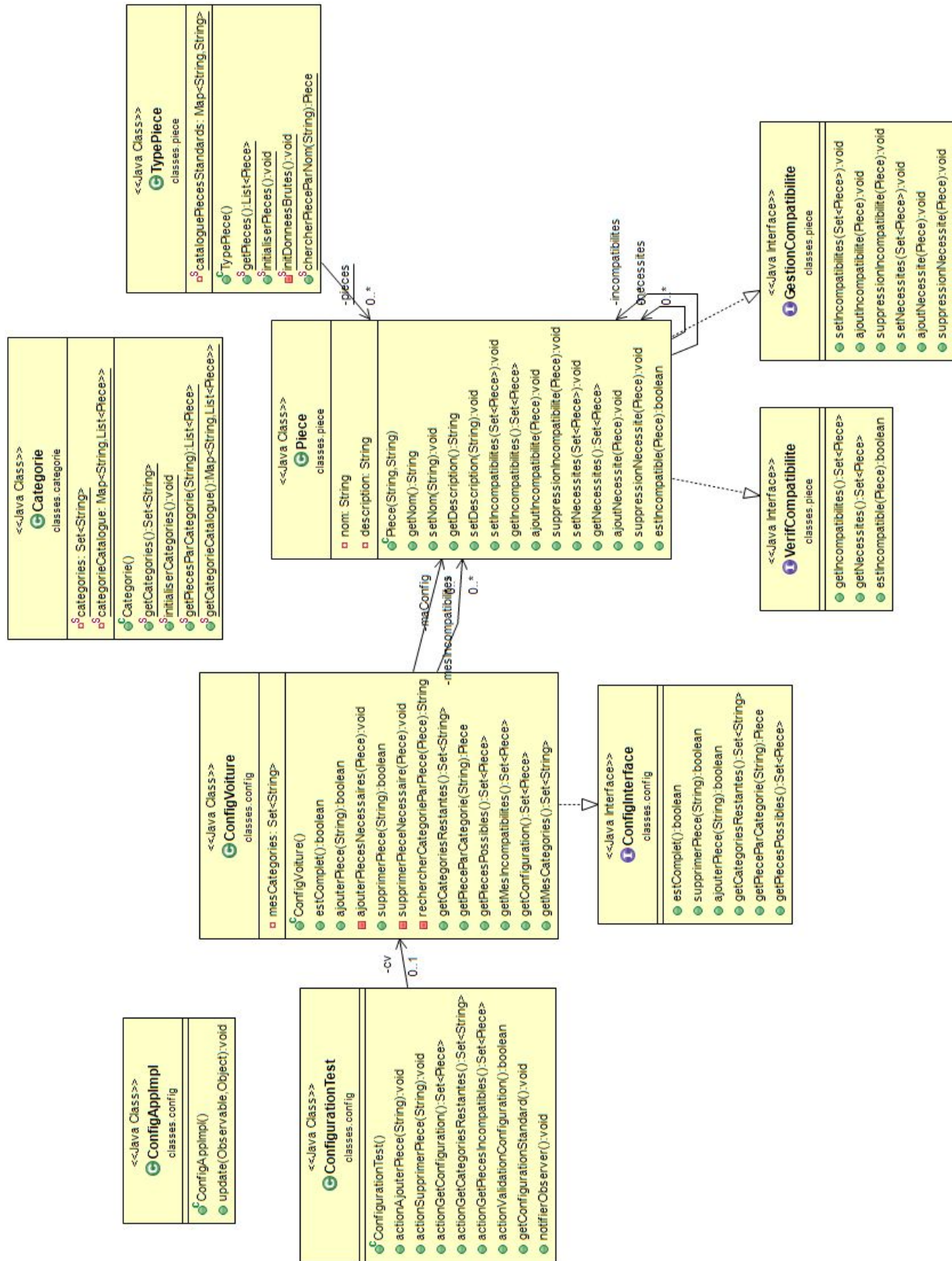
M1 MIAGE, groupe TP 1

Thomas GUESSANT

Charlotte LAURENSAN

Enseignant : Charles Quéguiner

Vue d'ensemble : architecture UML





Le diagramme ci-dessus représente le diagramme de classe de la version 1 du projet.

Objectifs

L'objectif de ce projet est de créer une application gérant une configuration de voiture appelée **Car Tailor**. En lien avec une interface graphique, l'utilisateur construit son propre véhicule en ajoutant une pièce pour chaque catégorie.

Lors de la création d'une voiture certaines pièces sont incompatibles et d'autres nécessaires, l'objectif de cette application est aussi de gérer l'ensemble des conditions existantes pour chaque pièce.

Voici les user stories que nous devons vérifier au cours du projet :

- #US1 : L'utilisateur peut afficher la liste des catégories afin d'en sélectionner une
- #US2 : L'utilisateur peut sélectionner une pièce d'une catégorie et l'insérer dans sa configuration
- #US3 : L'utilisateur peut supprimer n'importe quelle pièce de sa configuration
- #US4 : L'utilisateur peut savoir si sa configuration est valide (ou non)
- #US5 : L'administrateur peut ajouter de nouvelles incompatibilités ou nécessités pour une pièce

Explications du projet


Notre projet est un projet Maven permettant de récupérer les librairies automatiquement.

De plus, afin d'améliorer la qualité du code (comme éviter la duplication de code, respecter les règles de programmation, etc.), le code source a été analysé par Sonar Lint.

Gestion des pièces

Classes

La classe *Piece* permet de gérer la création d'une instance de pièce. Elle initialise un nom, une description, des pièces incompatibles (optionnel) et des pièces nécessaires (optionnel). L'administrateur peut, quant à lui, ajouter des incompatibilités ou des nécessités aux pièces (ref #US5). Une propriété peut être ajoutée à une *Piece*, elle est notamment utilisée pour donner une couleur aux pièces de la catégorie extérieure. Pour la version 2, une piece est une classe propre héritant des attributs de son type de piece (la catégorie), héritant directement de la classe mère *Piece*. Chaque piece possède un prix défini. Cependant les pièces extérieures peuvent entraîner un coût supplémentaire dû à la propriété une couleur.



La classe *TypePiece* est composée d'un ensemble de méthodes statiques ce qui rend l'instanciation de cette classe inutile. Elle regroupe, sous forme de liste, l'ensemble des pièces existantes du projet. Nous avons instancié les pièces une à une avant de leur ajouter des pièces incompatibles et nécessaires. Dès lors, on peut faire appel à la méthode "*TypePiece.initialiserPieces()*" créant toutes les pièces du projet.

Interfaces

L'interface *GestionCompatibilite* gère la compatibilité des pièces entre elles, donc l'ajout et la suppression des nécessités ou des incompatibilités.

L'interface *VerificationCompatibilite* renvoie les incompatibilités et les nécessités d'une pièce.

Gestion des catégories

La classe *Catégorie* est, elle aussi, composée de méthodes statiques. Seule cette classe gère les catégories et aucune interface ne l'implémente.

Les catégories sont les suivantes :

- "Engine" : le moteur du véhicule
- "Transmission" : la transmission de la voiture
- "Exterior" : le style extérieur
- "Interior" : le style intérieur

Un set est d'abord implémenté contenant les différents noms de catégories précédemment explicités. De plus, une map est créée avec pour clé les catégories et pour valeurs, la liste des pièces associées. Cette map a pour objectif de nous fournir un catalogue de pièces en fonction des catégories.

La méthode "*Categorie.initialiserCategorie()*" fait appel à la méthode de création des pièces ("*TypePiece.initialiserPieces()*") et associe chaque pièce à une catégorie.

Gestion de la configuration de voiture

Classes

La classe *ConfigurationTest* est la classe en lien avec l'utilisateur, il s'agit du point d'entrée de l'application. L'utilisateur peut créer une configuration et effectuer différentes actions que nous listerons ci-dessous. Elle est observée par la classe *ConfigAppImpl* qui, dans le cas où l'utilisateur modifie sa configuration, lui notifie d'un changement d'état. Voici ce que l'utilisateur a la possibilité d'effectuer dans cette classe :

- voir la liste des catégories, où aucune pièce n'a été ajouté, et en sélectionner une (ref #US1)
- ajouter une pièce dans sa configuration (ref #US2)

- supprimer une pièce de sa configuration (ref #US3)
- voir les pièces présentes dans sa configuration
- voir les pièces incompatibles à sa configuration
- valider sa configuration de voiture (ref #US4)
- connaître le prix de la configuration courante
- récupérer une configuration créée automatiquement
- récupérer les couleurs possibles d'une pièce extérieure et la modifier

La classe *ConfigApplImpl* est une classe observateur de l'ensemble des actions effectuées par l'utilisateur dans la classe *ConfigurationTest* présentée ci-dessus.

La classe *ConfigVoiture* est la classe où sont stockées les méthodes de gestion de pièces dans une configuration. Cette classe permet d'ajouter et de supprimer des pièces dans la configuration. En cas d'ajout d'une pièce, cette classe vérifie si il existe des pièces nécessaires à cette dernière. Si c'est le cas, les pièces nécessaires s'ajoutent automatiquement de même pour la suppression des pièces. Aussi, l'administrateur a la possibilité d'ajouter une propriété (avec des attributs) aux pièces. Après l'instanciation d'une configuration, la méthode "Categorie.initialiserCategorie()" doit être appelée.

Interfaces

L'interface *ConfigInterface*, est implémentée par la classe *ConfigVoiture*, elle reprend les méthodes de gestion des pièces.

Création des exceptions

Afin de faciliter la compréhension des erreurs, un ensemble d'exceptions a été créé. Elles permettent notamment de vérifier la validité des paramètres (les postconditions et préconditions).

Voici leur nom et une présentation de quelques cas pour lesquels elles sont appelées :


ParametreIncorrectException, comme son nom l'indique, est lancée si le paramètre d'une méthode n'est pas correct (vérification des préconditions). Par exemple :

- lors de l'ajout d'une pièce comportant le même nom qu'une autre pièce
- lors de la récupération de la liste de pièces d'une catégorie inexistante

ResultatIncorrectException est lancée lorsqu'une méthode ne renvoie pas de résultat ou le résultat attendu (vérification des postconditions). Par exemple :

- lors de la création d'une pièce, celle-ci n'est associée à aucune catégorie, c'est pourquoi on ne peut pas récupérer sa catégorie

ActionPieceInvalideException, est l'exception qui gère les actions faites sur la configuration de la voiture (la suppression ou l'ajout de pièces). Par exemple :

- 
- lors de l'ajout d'une pièce, elle ne peut pas être ajoutée dans le cas où elle est incompatible avec une autre pièce de la configuration
 - lors de l'ajout d'une pièce, cette dernière ne peut pas être ajoutée dans le cas où elle est déjà présente à la configuration
 - lors de la suppression d'une pièce, si celle-ci n'est pas dans la configuration de la voiture alors on ne peut pas la supprimer

Résultat de tests

Deux types de tests sont effectués dans notre projet :

- les tests unitaires vérifient les méthodes d'une classe de façon isolée
- les tests d'intégration vérifient le fonctionnement des méthodes en interaction avec les autres classes

Dans ce projet, l'ensemble des classes sont liées entre elles. Par exemple, lors de la création d'une configuration de voiture (*ConfigVoiture*), on doit initialiser les catégories (*Categorie*) initialisant elle-même le catalogue de pièces (*TypePiece*) pour gérer les pièces (*Piece*) de la voiture (voir diagramme de séquence ci-dessous).

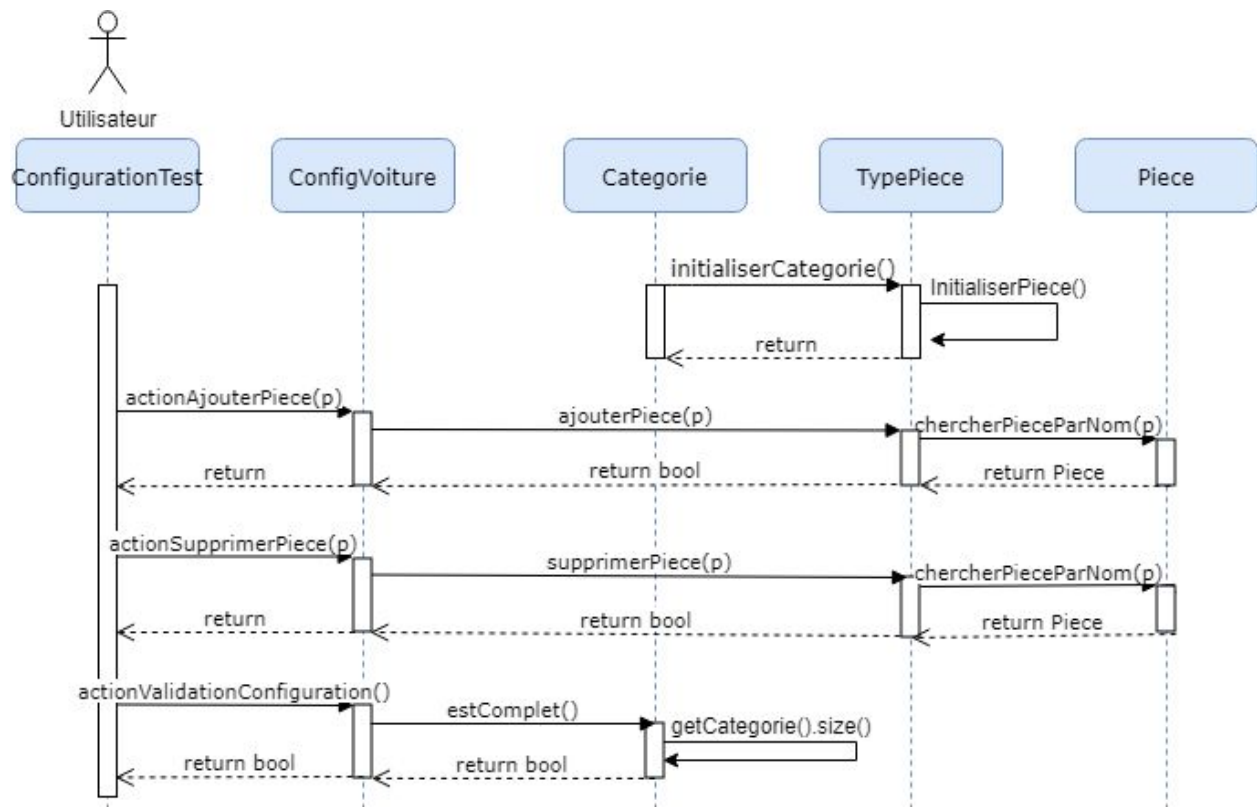














Diagramme de séquence synthétisant la création d'une configuration de voiture

L'initialisation des catégories ne doit être effectuée qu'une seule et unique fois dans une classe de tests, sinon cela provoque des erreurs. En effet si on ré-initialise les catégories, les pièces se créent à nouveau ce qui lance une Exception car la pièce existe déjà dans le catalogue de pièce.













Nous regardons le pourcentage de couverture de code afin d'avoir une idée sur les méthodes testées et celles encore en suspens. Suite à quoi, nous réalisons de nouveaux tests pour couvrir le plus de code possible.

Voici les différentes couvertures de code pour chaque classe pour la V1 :

<i>Piece</i>	▸  Piece.java	 100,0 %	216	0	216
<i>TypePiece</i>	▸  TypePiece.java	 99,0 %	306	3	309
<i>Categorie</i>	▸  Categorie.java	 98,4 %	179	3	182
<i>ConfigVoiture</i>	▸  ConfigVoiture.java	 100,0 %	293	0	293
<i>ConfigApplImpl</i>	▸  ConfigApplImpl.java	 100,0 %	30	0	30
	▸  ConfigurationTest.java	 100,0 %	63	0	63








A l'heure actuelle, plus de 98% du programme est testé pour la version 1 du projet.

Voici les différentes couvertures de code pour chaque classe pour la V2 :

<i>Piece</i>	▸  Piece.java	 100,0 %	370	0	370
<i>TypePiece</i>	▸  TypePiece.java	 98,9 %	277	3	280
<i>Categorie</i>	▸  Categorie.java	 98,4 %	179	3	182
<i>ConfigVoiture</i>	▸  ConfigVoiture.java	 77,2 %	429	127	556
<i>ConfigApplImpl</i>	▸  ConfigurationTest.java	 92,9 %	79	6	85
	▸  ConfigApplImpl.java	 100,0 %	30	0	30

Aussi, plus de 77% du programme est testé pour la version 2 du projet.

Comme dit précédemment on ne doit pas initialiser les catégories plus d'une fois, si l'ensemble des classes de tests sont exécutées en même temps (via le package) alors cela empêche le bon fonctionnement des tests unitaires. Voici une capture d'écran de cette situation :

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
ACO_201819	 16,0 %	825	4 344	5 169
src	 16,0 %	825	4 344	5 169
▸ classes.piece	 59,2 %	584	403	987
▸ classes.categorie	 98,4 %	179	3	182
▸ tests.GuessantLaurensan	 3,2 %	58	1 768	1 826
▸ exceptions	 33,3 %	4	8	12
▸ classes.config	 0,0 %	0	671	671

Cas d'erreur : exécution de JUnit Test sur le package