Université du Québec
à Trois-Rivières

TRAVAUX PRATIQUE 2

PRÉSENTÉ À

Monsieur Mourad Badri

POUR LE COURS

INF1007-Conception Logiciel

PAR

COMPAORE YANN DJAMEL

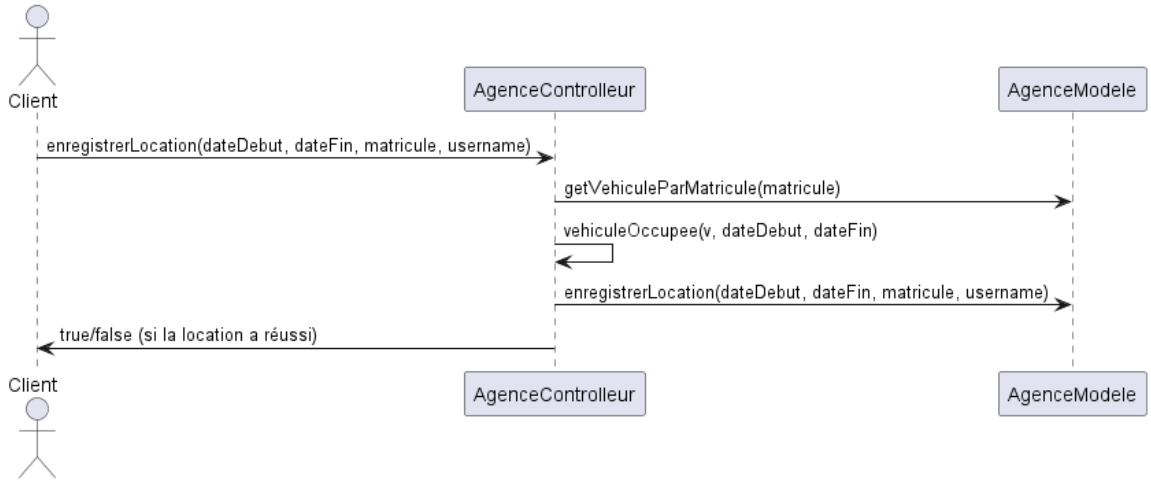
MOHAMED CAMARA

Partie 1 : Conception (partie équipe)

Les diagrammes sont disponibles en format.png dans le projet au cas où.

Diagrammes1 d'interaction issus des cas d'utilisations et explications

Diagramme d'interaction issus des cas d'utilisation 'Louer Véhicule'



Explications :

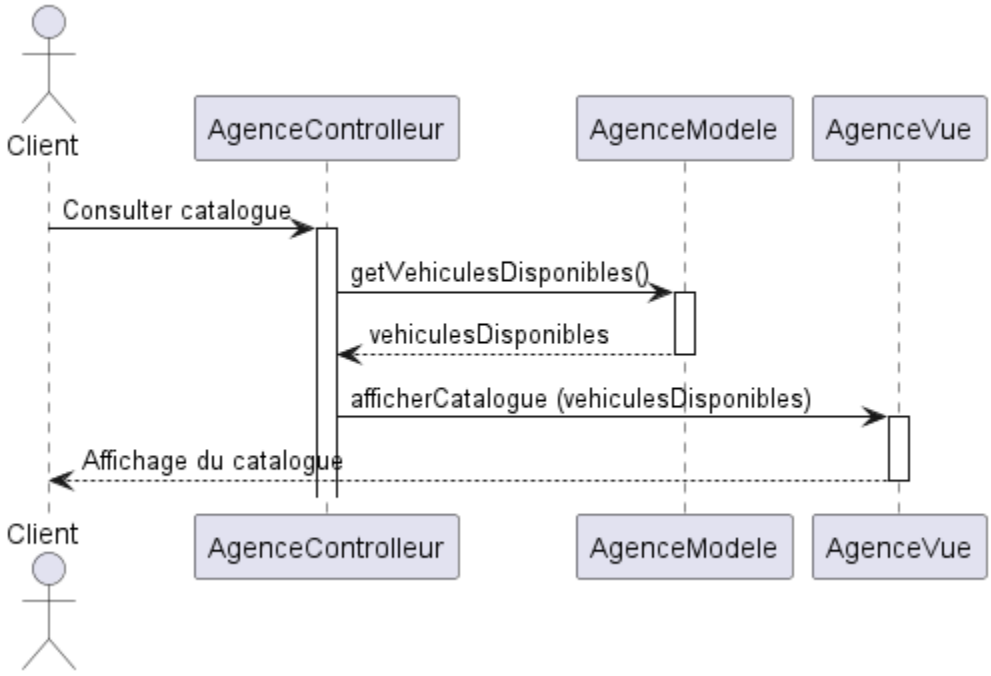
Les principales décisions concernant l'affectation des responsabilités aux classes sont les suivantes :

1. Utilisation du modèle MVC (Modèle-Vue-Contrôleur) : Le code suit le modèle de conception MVC, qui sépare les responsabilités entre les classes. Le modèle (AgenceModele) contient les données et la logique métier, la vue (AgenceVue) gère l'affichage et l'interaction utilisateur, et le contrôleur (AgenceControlleur) gère la communication entre le modèle et la vue.

2. Gestion des utilisateurs : Le code utilise plusieurs classes pour représenter différents types d'utilisateurs (Administrateur, Gestionnaire, Manager, Prepose, Client). Cela permet de définir des responsabilités spécifiques pour chaque type d'utilisateur et d'encapsuler les comportements liés aux rôles des utilisateurs.

3. Encapsulation des données : Les classes du modèle, comme Vehicule et Client, encapsulent les données et fournissent des méthodes pour accéder et modifier ces données. Cela permet de protéger les données et d'assurer leur intégrité.

Diagramme2 d'interaction issus des cas d'utilisation 'Consulter Catalogue'



Explications :

Les responsabilités ont été attribuées aux classes et aux acteurs en fonction des principes de la conception orientée objet. Les principales décisions d'affectation des responsabilités sont basées sur la cohésion et le couplage faible entre les classes.

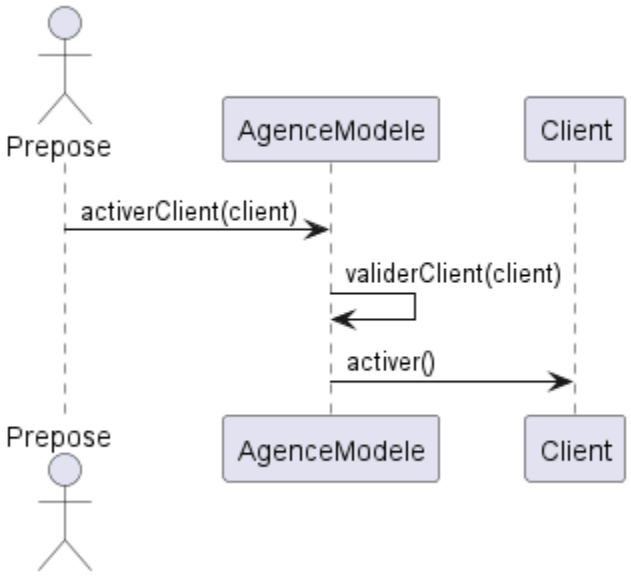
1. Le Client est l'acteur qui interagit avec le système pour consulter le catalogue. Il n'a pas de responsabilités particulières dans ce cas d'utilisation, car il ne fait que demander des informations.

2. Le Système est responsable de la gestion du catalogue. Il est chargé de fournir les informations sur les véhicules disponibles en utilisant la méthode **getListeVehicules()** de la classe Catalogue et d'afficher les résultats de la recherche de véhicules par marque en utilisant la méthode **getVehiculesParMarque()**.

3. La classe Catalogue est responsable de la gestion de la liste des véhicules disponibles. Elle contient une liste d'objets de type Vehicule et les méthodes pour ajouter, retirer et rechercher des véhicules par marque. Les méthodes de cette classe sont principalement des getters et des setters pour accéder et modifier la liste des véhicules.

4. La classe Véhicule contient les attributs tels que la marque, le modèle et le prix journalier. Elle n'est pas directement représentée dans le diagramme de séquence, mais elle est utilisée par la classe Catalogue pour gérer les informations sur les véhicules.

Diagramme3 d'interaction issus des cas d'utilisation 'Activer Client'(Gérer Client)



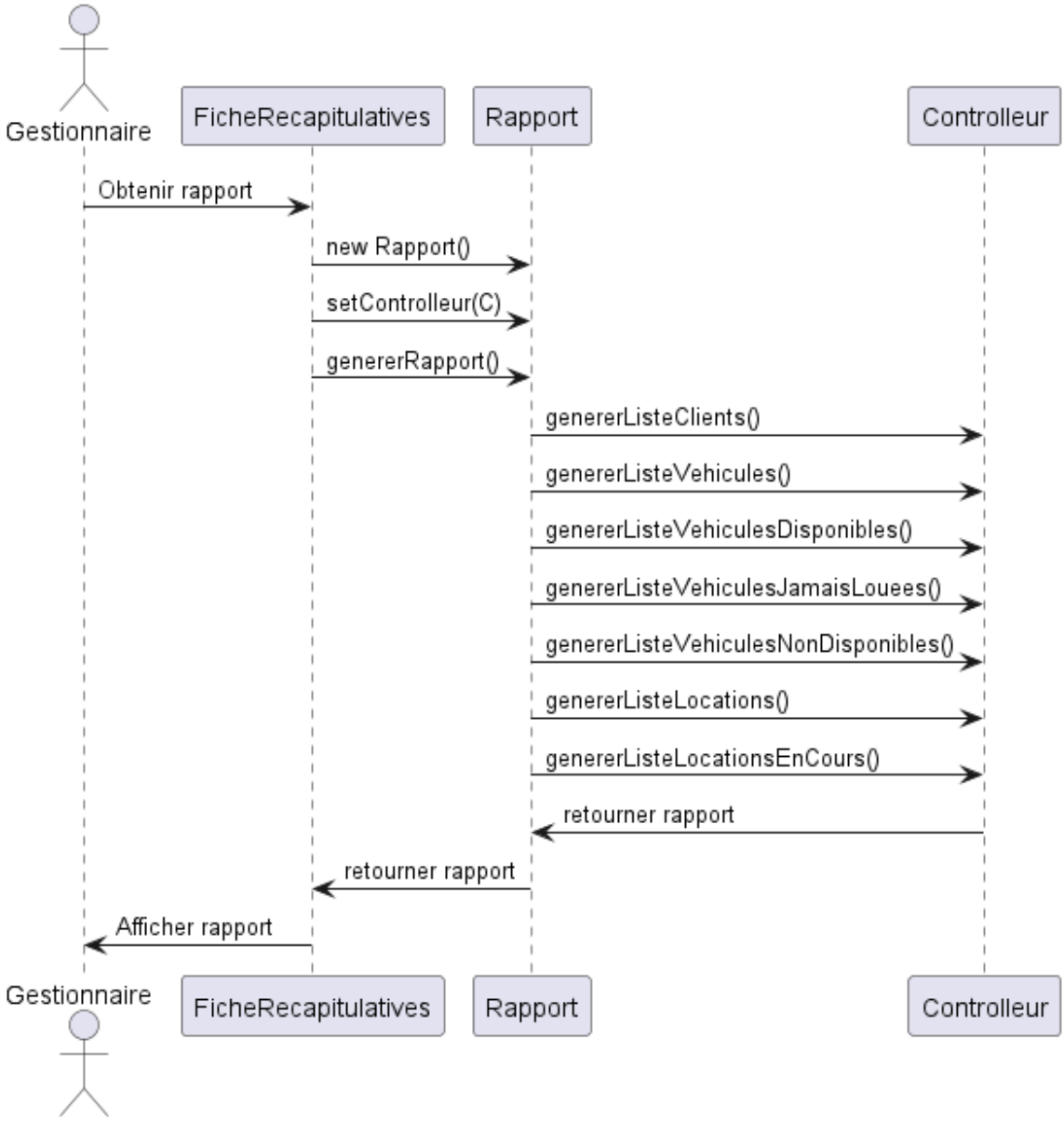
Explications :

Pour le cas d'utilisation "Activer Client", nous pouvons utiliser le pattern de conception "Commande". Nous pouvons considérer que le préposé est un "Invoker" qui prend en charge la demande de l'activation d'un client. Le Client peut être considéré comme un "Receiver" qui effectue l'action d'activation. La classe AgenceModele peut être considérée comme un "Client" qui contient toutes les informations nécessaires pour effectuer l'activation.

Ainsi, dans le diagramme de communication, le préposé envoie une commande d'activation de client à la classe AgenceModele, qui effectue l'action d'activation sur le client. La classe préposé peut également effectuer des vérifications pour s'assurer que l'activation est possible avant d'envoyer la commande à AgenceModele.

En outre, il est important de noter que la classe préposé est la seule autorisée à effectuer cette action, car elle est responsable de la sécurité et de l'authentification des Clients. Ainsi, la classe AgenceModele ne peut pas effectuer l'action d'activation sans l'approbation du préposé.

Diagramme4 d’interaction issus des cas d’utilisation ‘ObtenirRapport’



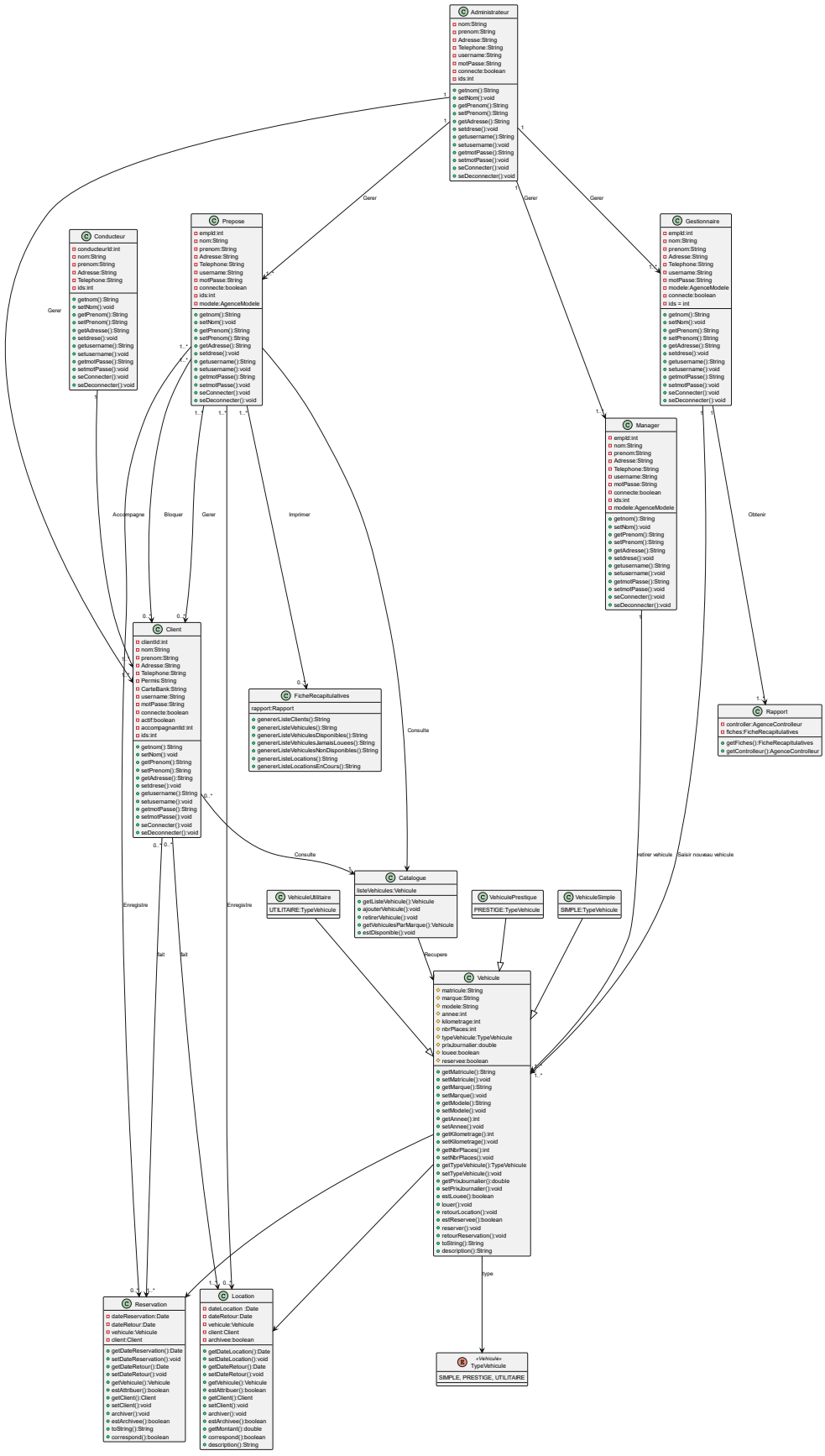
Explications :

On peut observer que le pattern de conception utilisé ici est le pattern MVC (Modèle-Vue-Contrôleur). Dans ce pattern, la classe **FicheRecapitulatives** correspond à la partie "modèle", qui contient les informations nécessaires pour générer le rapport. La classe **Rapport** correspond à la partie "contrôleur", qui orchestre la génération du rapport en appelant les méthodes appropriées de la

partie "modèle". Enfin, la classe **Gestionnaire** correspond à la partie "vue", qui demande le rapport et affiche les résultats.

On peut également noter que la classe **Controlleur** est utilisée comme une classe utilitaire pour faciliter la génération de listes de véhicules et de locations, ce qui permet de réutiliser le même code à plusieurs endroits dans l'application.

Diagramme de classes de conception (DCC).


Administrateur
nom:String
prenom:String
Adresse:String
Telephone:String
username:String
motPasse:String
connecte:boolean
ids:int
getnom():String
setNom():void
getPrenom():String
setPrenom():String
getAdresse():String
setdrese():void
getusername():String
setusername():void
getmotPasse():String
setmotPasse():void
seConnecter():void
seDeconnecter():void
Gerer
Conducteur
conducteurId:int
nom:String
prenom:String
Adresse:String
Telephone:String
ids:int
getnom():String
setNom():void
getPrenom():String
setPrenom():String
getAdresse():String
setdrese():void
getusername():String
setusername():void
getmotPasse():String
setmotPasse():void
seConnecter():void
seDeconnecter():void
Prepose
emplId:int
nom:String
prenom:String
Adresse:String
Telephone:String
username:String
motPasse:String
connecte:boolean
ids:int
modele:AgenceModele
getnom():String
setNom():void
getPrenom():String
setPrenom():String
getAdresse():String
setdrese():void
getusername():String
setusername():void
getmotPasse():String
setmotPasse():void
seConnecter():void
seDeconnecter():void
Gestionnaire
emplId:int
nom:String
prenom:String
Adresse:String
Telephone:String
username:String
motPasse:String
modele:AgenceModele
connecte:boolean
ids = int
getnom():String
setNom():void
getPrenom():String
setPrenom():String
getAdresse():String
setdrese():void
getusername():String
setusername():void
getmotPasse():String
setmotPasse():void
seConnecter():void
seDeconnecter():void
Manager
emplId:int
nom:String
prenom:String
Adresse:String
Telephone:String
username:String
motPasse:String
connecte:boolean
ids:int
modele:AgenceModele
getnom():String
setNom():void
getPrenom():String
setPrenom():String
getAdresse():String
setdrese():void
getusername():String
setusername():void
getmotPasse():String
setmotPasse():void
seConnecter():void
seDeconnecter():void
Accompagne
Bloquer
Gerer
Imprimer
Consulte
Obtenir
Client
clientId:int
nom:String
prenom:String
Adresse:String
Telephone:String
Permis:String
CarteBank:String
username:String
motPasse:String
connecte:boolean
actif:boolean
accompagnantId:int
ids:int
getnom():String
setNom():void
getPrenom():String
setPrenom():String
getAdresse():String
setdrese():void
getusername():String
setusername():void
getmotPasse():String
setmotPasse():void
seConnecter():void
seDeconnecter():void
FicheRecapitulatives
rapport:Rapport
genererListeClients():String
genererListeVehicules():String
genererListeVehiculesDisponibles():String
genererListeVehiculesJamaisLouees():String
genererListeVehiculesNonDisponibles():String
genererListeLocations():String
genererListeLocationsEnCours():String
Rapport
controller:AgenceController
fiches:FicheRecapitulatives
getFiches():FicheRecapitulatives
getController():AgenceController
Consulte
Catalogue
listeVehicules:Vehicule
getListeVehicule():Vehicule
ajouterVehicule():void
retirerVehicule():void
getVehiculesParMarque():Vehicule
estDisponible():void
VehiculeUtilitaire
UTILITAIRE:TypeVehicule
VehiculePrestigue
PRESTIGE:TypeVehicule
VehiculeSimple
SIMPLE:TypeVehicule
retirer vehicule
Saisir nouveau vehicule
Enregistre
Enregistre
fait
fait
Recupere
Vehicule
matricule:String
marque:String
modele:String
annee:int
kilometrage:int
nbrPlaces:int
typeVehicule:TypeVehicule
prixJournalier:double
louee:boolean
reservee:boolean
getMatricule():String
setMatricule():void
getMarque():String
setMarque():void
getModele():String
setModele():void
getAnnee():int
setAnnee():void
getKilometrage():int
setKilometrage():void
getNbrPlaces():int
setNbrPlaces():void
getTypeVehicule():TypeVehicule
setTypeVehicule():void
getPrixJournalier():double
setPrixJournalier():void
estLouee():boolean
louer():void
retourLocation():void
estReservee():boolean
reserver():void
retourReservation():void
toString():String
description():String
type
Reservation
dateReservation:Date
dateRetour:Date
vehicule:Vehicule
client:Client
getDateReservation():Date
setDateReservation():void
getDateRetour():Date
setDateRetour():void
getVehicule():Vehicule
estAttribuer():boolean
getClient():Client
setClient():void
archiver():void
estArchivee():boolean
toString():String
correspond():boolean
Location
dateLocation :Date
dateRetour:Date
vehicule:Vehicule
client:Client
archivee:boolean
getDateLocation():Date
setDateLocation():void
getDateRetour():Date
setDateRetour():void
getVehicule():Vehicule
estAttribuer():boolean
getClient():Client
setClient():void
archiver():void
estArchivee():boolean
getMontant():double
correspond():boolean
description():String
«Verticale»
TypeVehicule
SIMPLE, PRESTIGE, UTILITAIRE

Partie 2 : Implémentation (partie individuelle)

Cas d’utilisateur : Louer Véhicule ,Diagramme1 //COMPAORE YANN

Cas d’utilisateur : Enregistrer Rapport ,Diagramme4//COMPAORE YANN

Cas d’utilisateur : Consulter Rapport ,Diagramme2 //Mohamed Camara

Cas d’utilisateur : Gérer Client ,Diagramme3//Mohamed Camara

Partie 3 : Plan d'intégration (partie équipe)

Choix d’une stratégie d’intégration parmi celles étudiées en cours (celle qui nous semble la plus appropriée).

Une stratégie d'intégration ascendante est plus adaptée. Cette stratégie consiste en une intégration couche par couche des composants. Commencez par la couche la plus basse et progressez progressivement.

L'avantage de cette approche est que les erreurs peuvent être détectées tôt et corrigées rapidement. En fait, les erreurs sont déjà détectées lorsque les couches inférieures sont intégrées, de sorte que la cause de l'erreur peut être rapidement identifiée. De plus, cette stratégie nous permet également de tester l'interface de chaque composant avec ses voisins dès le début de l'intégration.

Ordre des opérations de test des classes :

1. VehiculeSimple
2. VehiculePrestique
3. VehiculeUtilitaire
4. Véhicule
5. Réservation
6. Location
7. Client
8. Conducteur
9. Administrateur
10. préposé
11. Gestionnaire
12. Catalogue
13. FicheRecapitulatives
14. Rapport