📘 Cahier des Charges Global : FleetManager

1. Vision du Projet

FleetManager est une application de gestion de flotte de location de véhicules destinée à une agence de location ("LocaFast"). L'objectif est de remplacer les processus manuels par une solution logicielle robuste, orientée objet, capable de gérer les véhicules, les clients, les locations et la persistance des données.

Architecture Technique :

- **Langage :** Java (JDK 17 ou supérieur recommandé).
- **Interface :** Swing (Graphique) et Console (pour les tests initiaux).
- **Persistance :** Fichiers textes (CSV/TXT) pour la sauvegarde/chargement.
- **Pattern :** Architecture en couches (Model, Service/Controller, View).

2. Modèle de Données (Le "Quoi") - Package `fr.fleetmanager.model`

Ce module définit les objets manipulés par l'application.

2.1 Hiérarchie des Véhicules

Tous les véhicules partagent des caractéristiques communes mais ont des règles de tarification spécifiques.

- **Classe Abstraite** `Vehicule` (✅ Fait)
  - **Rôle :** Classe parente, définit le contrat commun.
  - **Attributs (tous** `private` **) :**
    - `id` (int) : Identifiant unique, généré automatiquement via un compteur statique.
    - `marque` (String).
    - `modele` (String).
    - `kilometrage` (double) : Kilométrage actuel.
    - `tarifJournalier` (double) : Prix de base par jour.
  - **Méthodes Clés :**
    - Constructeur complet.
    - Getters/Setters (avec validation : km ne peut pas diminuer).
    - `public abstract double calculerPrixLocation(int jours)` : Méthode polymorphique.
    - `toString()` : Description textuelle.
- **Classe** Concrète `Voiture` (hérite de `Vehicule`) (✅ Fait)
  - **Attribut Spécifique :** `nbPlaces` (int).
  - **Logique Métier (** `calculerPrixLocation` **) :** Prix = `tarifJournalier` * `jours` .
- **Classe Concrète** `Camion` **(hérite de** `Vehicule` **)** (✅ Fait)

- **Attribut Spécifique :** `volume` (double) en m^3.
- **Logique Métier (** `calculerPrixLocation` **) :** Prix = `(tarifJournalier` * jours) `+ (volume * 10)`.

2.2 Les Clients (⬜ À faire)

- Classe `Client`
  - **Attributs :**
    - `id` (int) : Unique (compteur statique).
    - `nom` (String).
    - `prenom` (String).
    - `numeroPermis` (String).
    - `vehiculeLoue` (`Vehicule`) : Référence vers le véhicule actuellement loué (peut être `null`).
  - **Règles :**
    - Un client ne peut avoir qu'un seul véhicule en location à la fois.

3. Logique Métier et Services (Le "Comment") - Package `fr.fleetmanager.service`

Ce module gère la logique de l'application et les manipulations de données.

- Classe `Agence` **(Le Contrôleur Principal) (🎯 Ta mission actuelle)**
  - **Rôle :** Point d'entrée pour la gestion du parc et des clients.
  - **Attributs :**
    - `parc` (`ArrayList<Vehicule>`) : Liste de tous les véhicules.
    - `clients` (`ArrayList<Client>`) : Liste de tous les clients enregistrés.
  - **Fonctionnalités Requises :**
    1. **Gestion Véhicules :**
       - `ajouterVehicule(Vehicule v)` : Ajoute au parc.
       - `supprimerVehicule(Vehicule v)` : Retire du parc.
       - `getVehiculeById(int id)` : Recherche par ID.
       - `getVehiculesDisponibles()` : Retourne la liste des véhicules non loués.
    2. **Gestion Clients :**
       - `ajouterClient(Client c)`.
       - `getClientByPermis(String permis)`.
    3. **Gestion Locations (Cœur du Métier) :**
       - `louerVehicule(Client c, Vehicule v)` :
         - Vérifie si le véhicule est disponible.
         - Vérifie si le client n'a pas déjà une location.
         - Associe le véhicule au client.

    - Lève une exception `LocationException` en cas d'erreur.
  - `rendreVehicule(Client c)` :
    - Libère le véhicule associé au client.
    - Met à jour le kilométrage (paramètre optionnel).
    - Retourne le prix total à payer.
4. **Reporting :**
  - `calculerRevenuPotentiel()` : Somme des locations journalières de tout le parc.

4. Persistance (Sauvegarde) - Package `fr.fleetmanager.persistence`

- Classe `FileManager`
  - Méthode `sauvegarderDonnees(Agence agence, String filename)` :
    - Écrit l'état des véhicules et des clients dans un fichier texte.
    - Format suggéré : CSV (ex: `TYPE;ID;MARQUE;MODELE...`).
  - Méthode `chargerDonnees(String filename)` :
    - Lit le fichier et reconstruit les objets `Agence`, `Vehicule`, `Client` en mémoire au démarrage.

5. Gestion des Erreurs - Package `fr.fleetmanager.exception`

Créer des exceptions personnalisées pour gérer proprement les erreurs métier.

- Classe `LocationException` (hérite de `Exception`)
  - Utilisée quand une location est impossible (véhicule indisponible, client a déjà loué, etc.).
- Classe `VehiculeIntrouvableException`
  - Utilisée lors des recherches infructueuses.

6. Interface Graphique (La "Vue") - Package `fr.fleetmanager.ui`

L'interface doit permettre à l'utilisateur d'interagir avec le système `Agence`.

- Fenêtre Principale (`MainFrame`)
  - Hérite de `JFrame`.
  - Utilise un `BorderLayout`.
- Zone Centrale :
  - `JTable` ou `JList` affichant la liste des véhicules (Marque, Modèle, Statut, Tarif).
- Barre d'Outils / Menus :
  - Boutons : "Ajouter Véhicule", "Nouveau Client", "Louer", "Rendre", "Sauvegarder".
- Dialogues (`JDialog` ou `JOptionPane`) :
  - Formulaire d'ajout de véhicule (choix Voiture/Camion via `JComboBox`).
  - Formulaire de location (Sélectionner un client, sélectionner un véhicule).