ARCHITECTURE TECHNIQUE ET LOGIQUE DE NOTRE SYSTEME

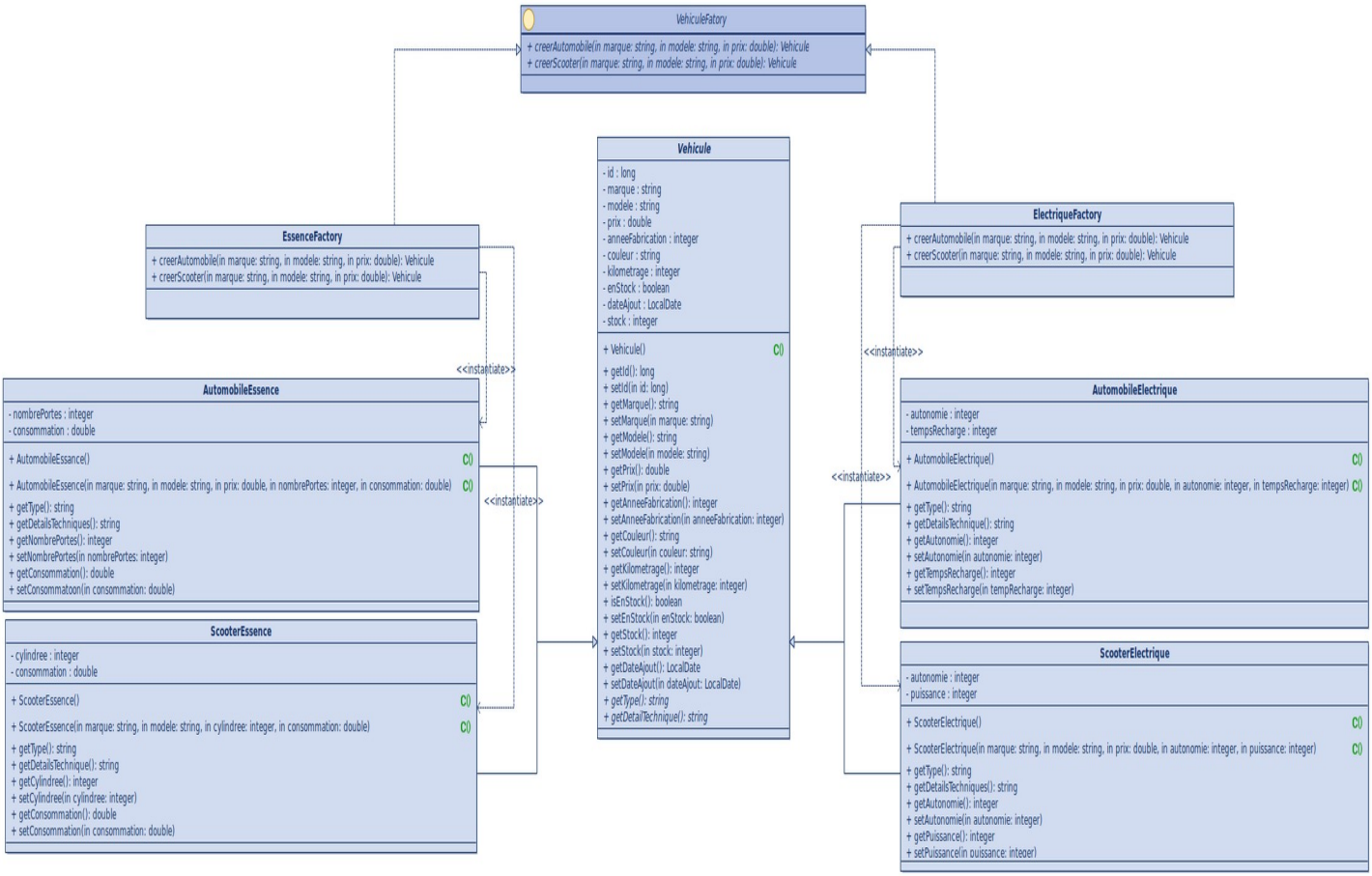
ARCHITECTURE LOGIQUE :

ARCHITECTURE TECHNIQUE

SOLUTIONS CONCEPTUELLES AVEC EXPLICATIONS

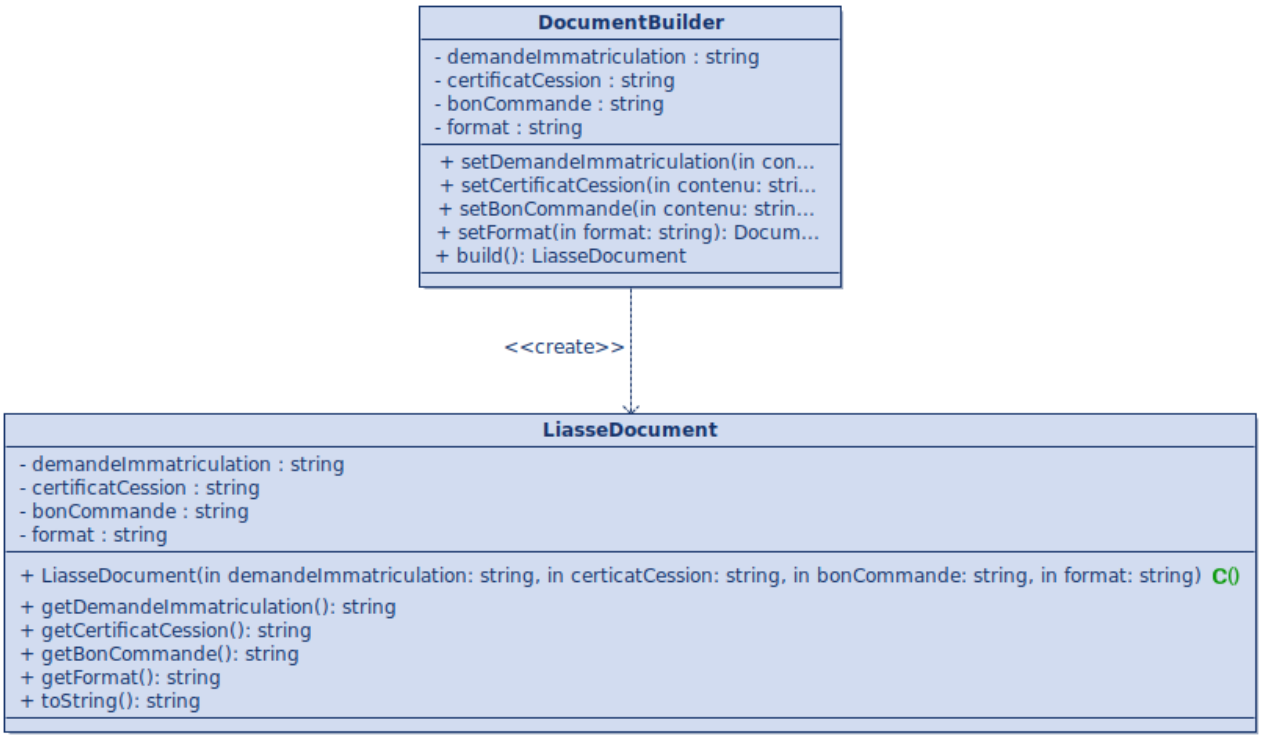
1 - Construire les objets du domaine (automobile à essence ou électrique, scooter à essence ou électrique, etc.) : **Abstract Factory**

Explications :



1. **VehiculeFactory** : Interface abstraite définissant les méthodes de création
2. **ElectriqueFactory** : Crée des véhicules électriques avec paramètres par défaut
3. **EssenceFactory** : Crée des véhicules essence avec paramètres par défaut
4. **AutomobileElectrique**, **ScooterElectrique** , **AutomobileEssence**, **ScooterEssence** : Sont les produit créées

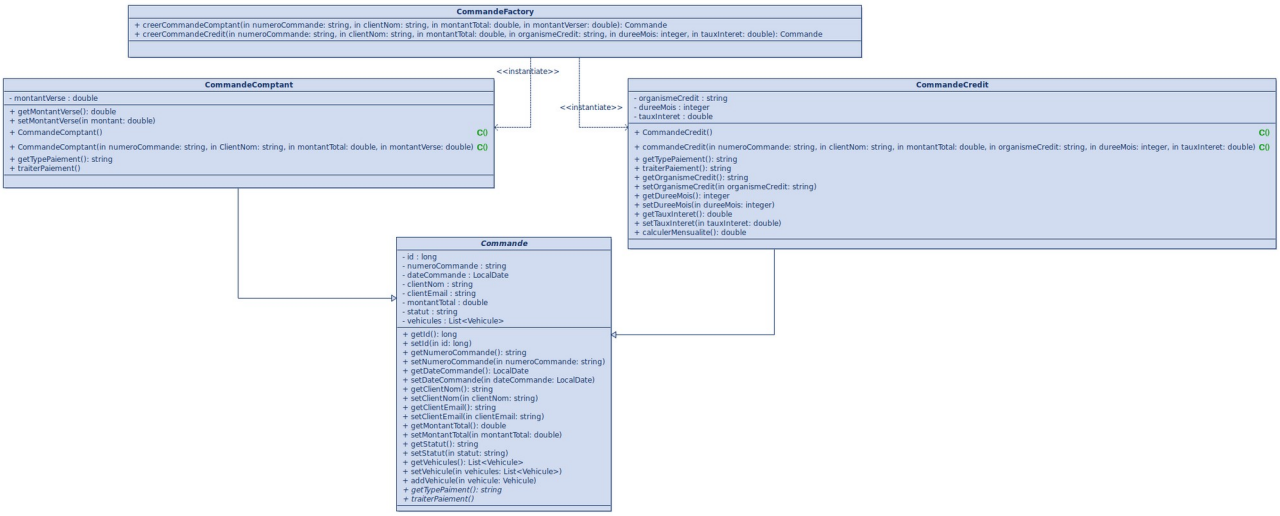
2 - Construire les liasses de documents nécessaires en cas d'acquisition d'un véhicule : **Builder**



Explications :

1. **DocumentBuilder** : Le Builder qui permet de construire étape par étape une liasse de documents.
    - Contient les attributs privés pour chaque document
    - Méthodes **setX()** qui retournent **DocumentBuilder**
    - Méthode **build()** qui valide et construit l'objet **final**
2. Classe **LiasseDocument** : Le produit final qui contient tous les documents assemblés.
    - Constructeur qui prend tous les paramètres
    - Représente l'objet complexe créé

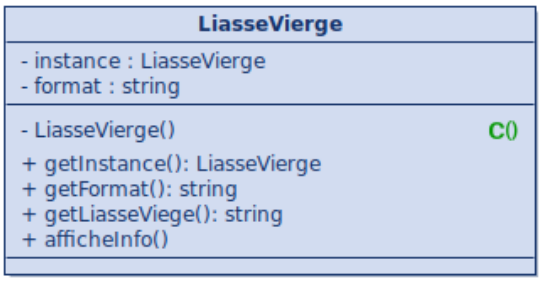
3 - Créer les commandes : **Factory Method**



Explications

1. **CommandeFactory :** Fabrique qui crée les commandes avec initialisation complète Méthodes de création spécifiques pour chaque type de commande
2. **Commande :** la classe abstraite dont les sous –classes seront instanciées
3. **CommandeComptant et CommandeCredit :** les sous classes concrètes à instancier

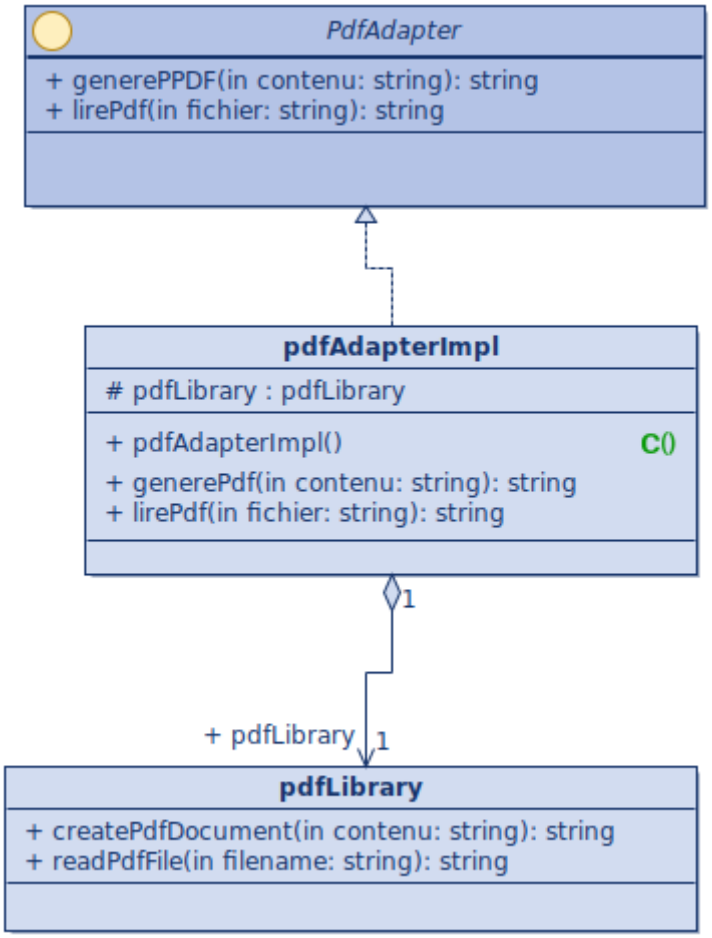
4 - Créer la liasse vierge de documents : **Singleton**



Explications

1. **Constructeur privé** : Empêche l'instanciation directe
2. **Instance statique** : Variable instance pour stocker l'unique instance
3. **Méthode getInstance()** : Point d'accès global synchronisé

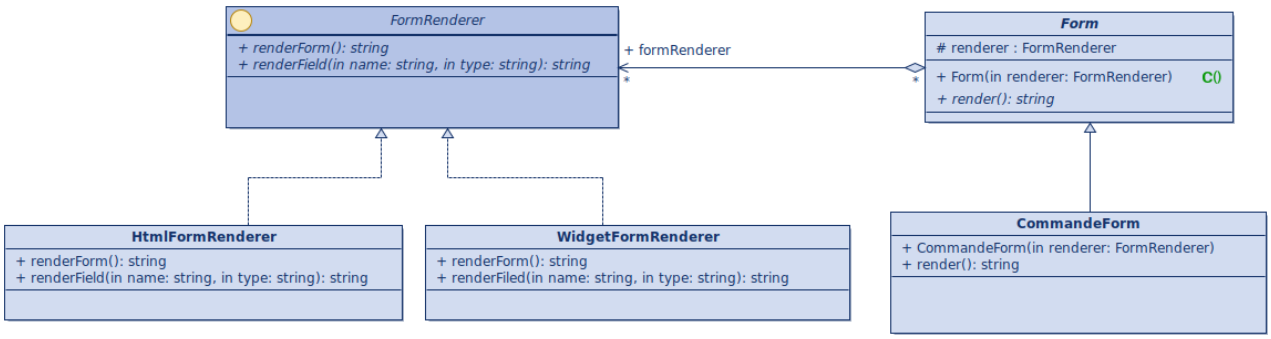
5 - Gérer des documents PDF : **Adapter**



Explications

1. Interface **PdfAdapter** : Définit les méthodes genererPdf() et lirePdf() en français.
2. Classe **PdfLibrary** : La classe existante avec ses méthodes originales en anglais (createPdfDocument() et readPdfFile()).
3. Classe **PdfAdapterImpl** : Implémente **PdfAdapter** et contient une **instance** de **PdfLibrary**

6 - Implanter des formulaires HTML ou à l'aide de widgets : **Bridge**

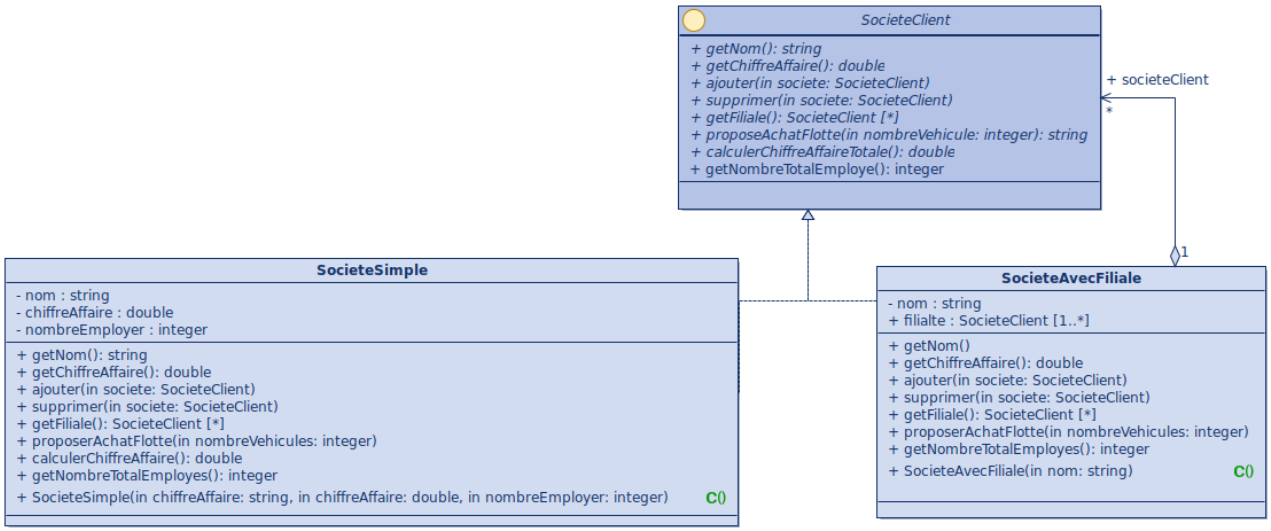


Explications :

1. **Interface FormRenderer** : Définit les méthodes de rendu des formulaires.
2. **Implémentations concrètes** : HtmlFormRenderer et WidgetFormRenderer implémentent l'interface FormRenderer.
3. **Classe abstraite Form** : Contient une référence à un FormRenderer et définit la méthode abstraite render().
4. **Classe concrète CommandeForm** : Hérite de Form et implémente la méthode render() spécifique aux formulaires de commande.

NB : Cela permet de changer indépendamment les types de formulaires et les moteurs de rendu.

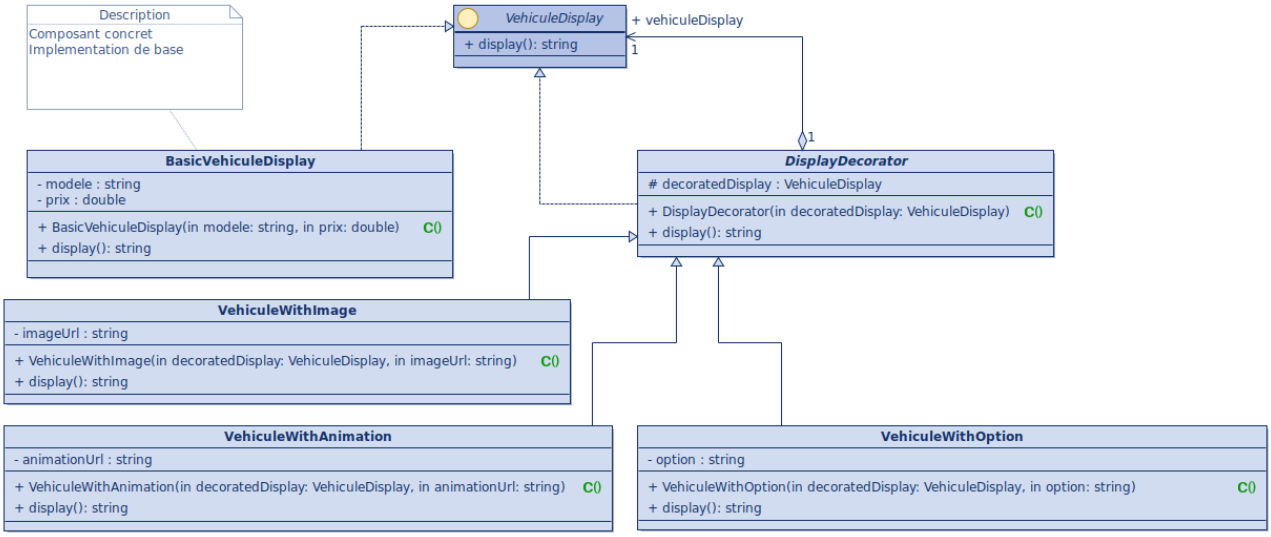
7 - Représenter les sociétés clientes : **Composite**



Explications :

1. **Interface SocieteClient** : Le composant commun qui définit l'interface pour tous les éléments.
2. **Classe** SocieteSimple : La feuille (Leaf) qui représente une société individuelle sans filiales.
3. **Classe** SocieteAvecFiliales : Le composite qui peut contenir d'autres sociétés (feuilles ou composites).

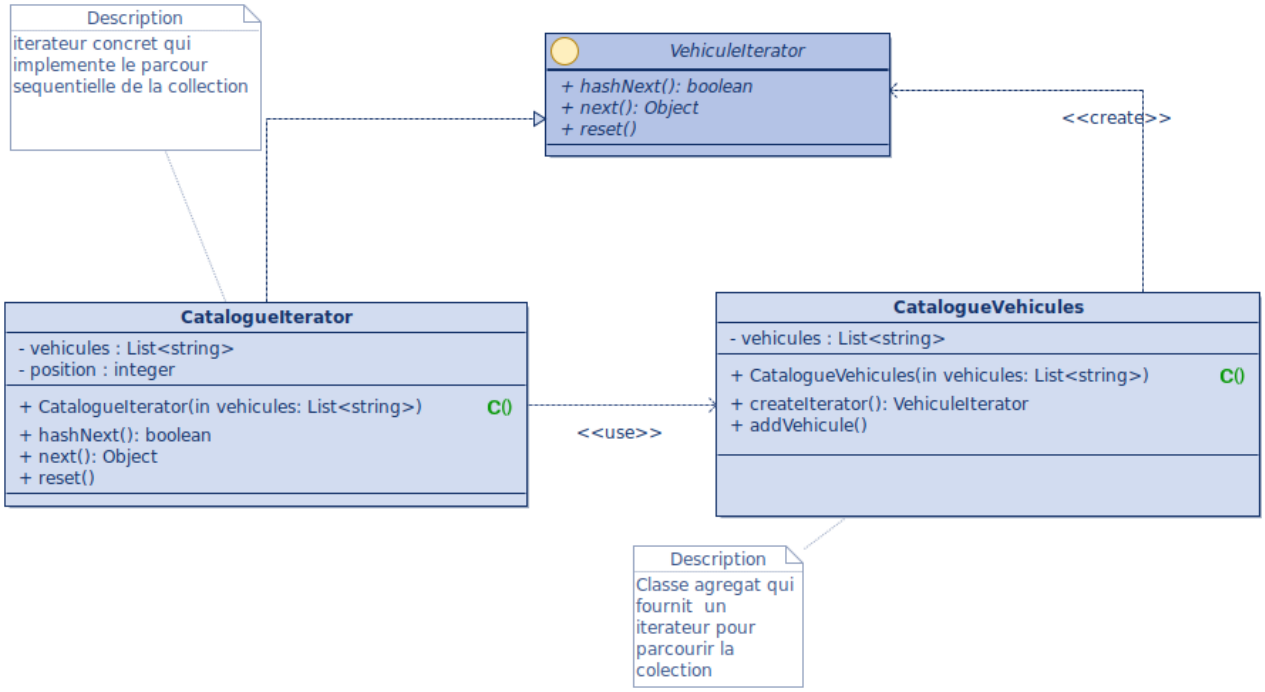
8 - Afficher les véhicules du catalogues : **Decorator**



Explications :

1. **Interface VehiculeDisplay** : Interface commune pour tous les composants
2. **Classe BasicVehiculeDisplay** : Implémentation de base (composant concret)
3. **Classe abstraite DisplayDecorator** : Décorateur abstrait qui maintient une référence vers le composant décoré
4. Décorateurs concrets :
    - **VehiculeWithImage** : Ajoute une image
    - **VehiculeWithOptions** : Ajoute des options
    - **VehiculeWithAnimation** : Ajoute une animation

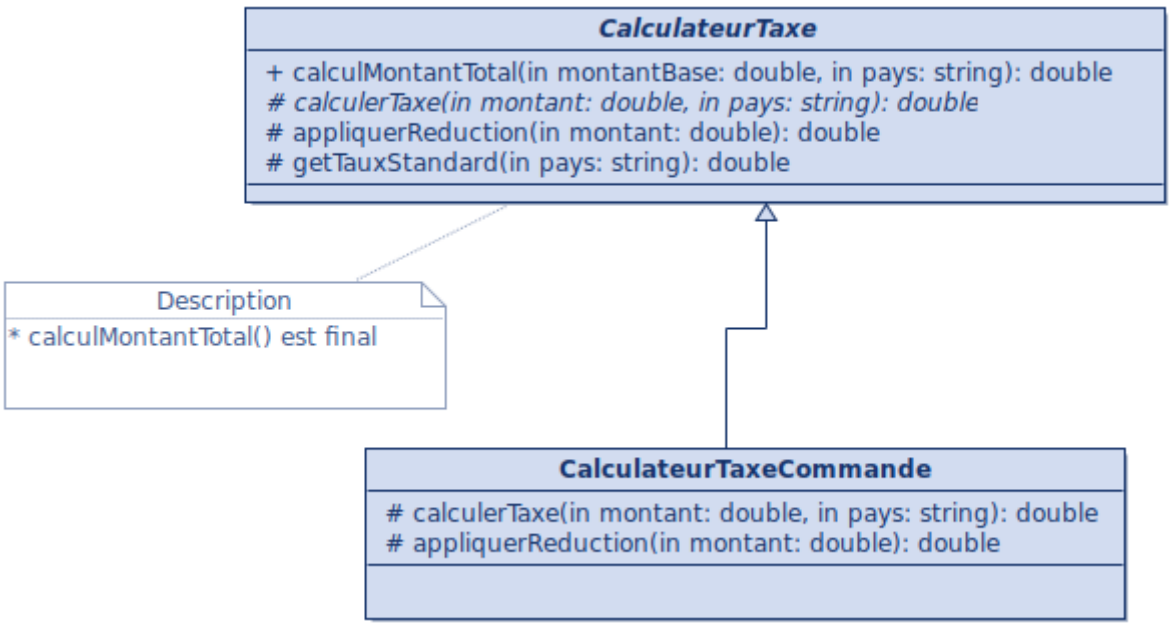
9 - Retrouver séquentiellement les véhicules du catalogue : **Iterator**



Explications :

1. **VehiculeIterator** : Interface définissant les opérations d'itération
2. **CatalogueIterator** : Implémentation concrète de l'itérateur
3. **CatalogueVehicules** : Collection/Agrégat qui fournit l'itérateur

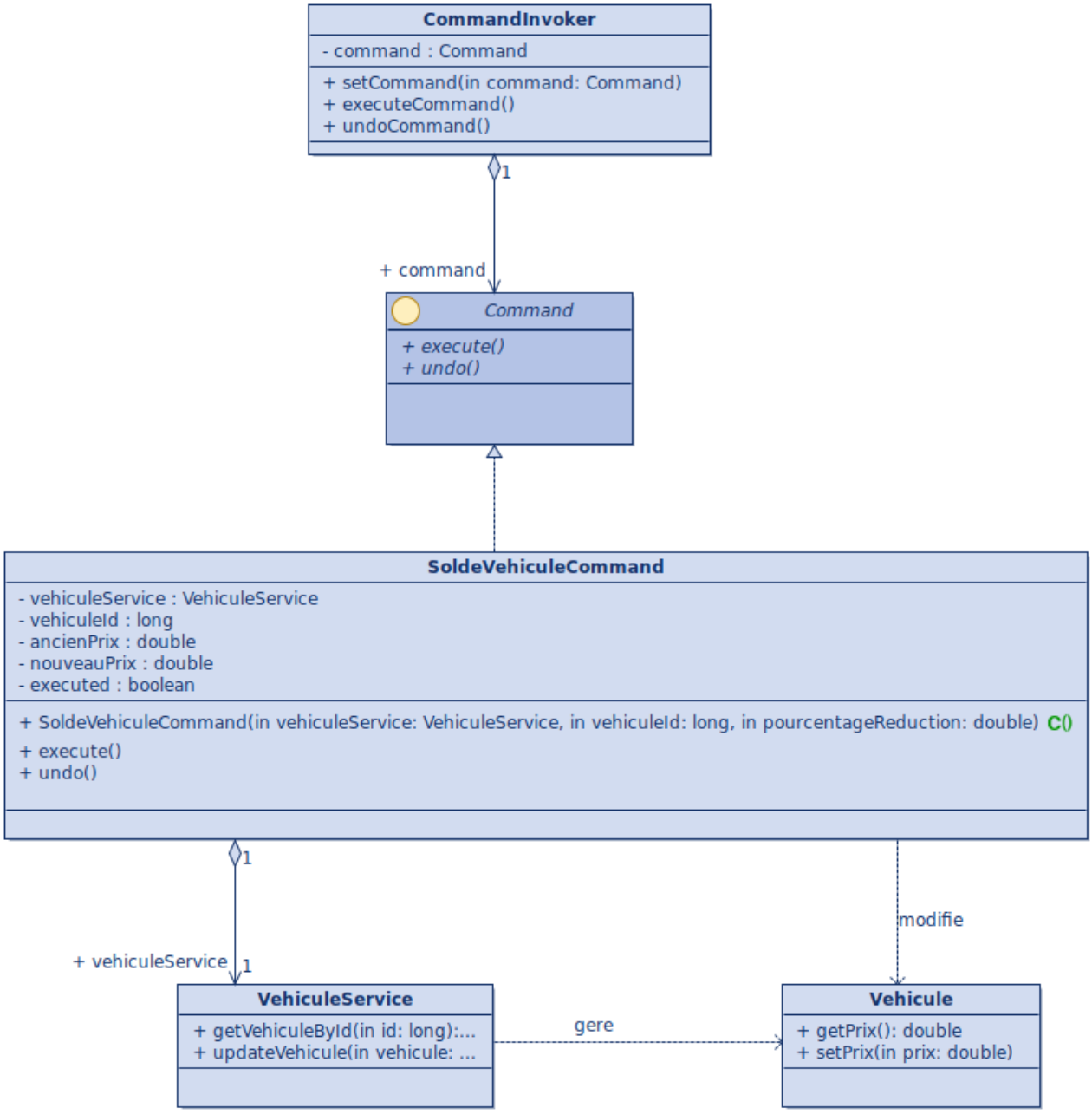
10 - Calculer le montant d’une commande : **Template Method**



Explications :

1. Classe abstraite **CalculateurTaxe** : Définit le squelette de l'algorithme
2. Méthode template **calculerMontantTotal()** : Méthode final qui orchestre les étapes
3. **Méthodes protégées** : Étapes de l'algorithme que les sous-classes peuvent redéfinir
4. **CalculateurTaxeCommande** : Surcharge les méthodes de l'algorithme :
   - **calculerTaxe()** : Ajoute une taxe supplémentaire pour la France sur gros montants
   - **appliquerReduction() :** Ajoute une réduction fixe supplémentaire
5. **getTauxStandard()** : Méthode helper réutilisable par toutes les sous-classes
6. **appliquerReduction()** : Implémentation par défaut qui peut être étendue

11 - Solder les véhicules restés en stock pendant une longue durée : **Command**



Explications :

1. **Interface Command**
   - Contrat minimal pour toutes les commandes

- Deux méthodes obligatoires :
  - **execute()** : exécute l'action
  - **undo()** : annule l'action

2. **Commande concrète SoldeVehiculeCommand**
   - Stocke l'état :
     - **ancienPrix** : prix avant solde
     - **nouveauPrix** : prix après solde
     - **executed** : flag pour éviter double exécution
   - Logique métier :
     - Dans le constructeur : **calcule les prix**
     - **execute()** : applique la réduction
     - **undo()** : restaure l'ancien prix
3. **Invoker CommandInvoker :**
   - **Ne connaît pas le détail** : Traite toutes les commandes de la même façon
   - **Simple délégation** : Appelle juste execute() ou undo()