DOCUMENTATION DE LOGICPC

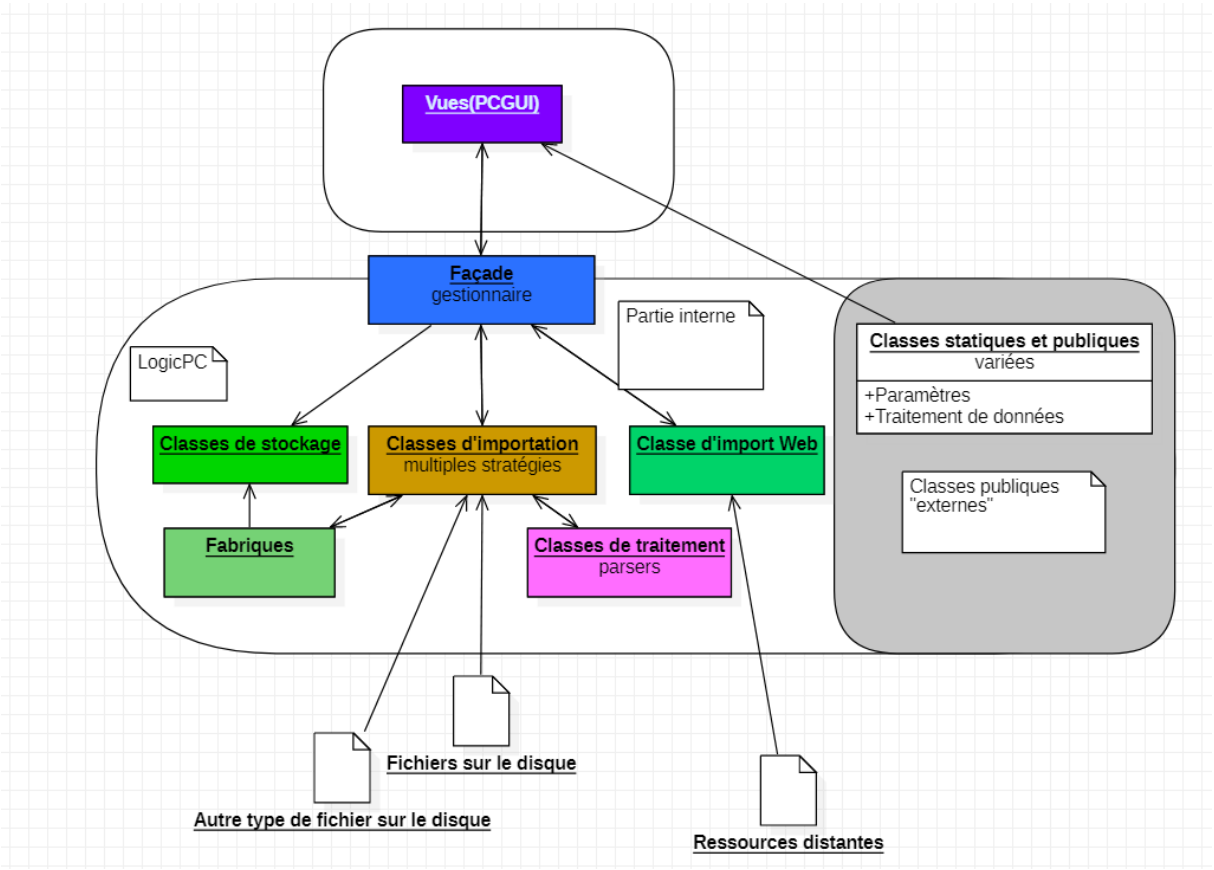
SOMMAIRE

TABLE DES MATIERES

1) DESCRIPTION DE L'ARCHITECTURE

LogicPC est la partie modèle du projet PcParted. Sous forme d'une bibliothèque de classes elle a pour responsabilité tous les traitements de données hors la gestion visuelle et la persistance complète (partielle ici).

On peut séparer LogicPC en 2 types de classes :
- Intégré : La classe est quasi-entièrement en attribut *internal* et n'est prévue que pour fonctionner à l'intérieur de la bibliothèque de classes
- Externe : La classe est publique et/ou statique. Elle peut être librement utilisée par les vues (surtout des classes de traitement et le gestionnaire).



Dans ce diagramme d'architecture simplifié, on peut voir les vues qui interagissent avec une façade. C'est la seule classe permettant un lien entre les classes « intégrées » à la bibliothèque et l'extérieur. Cette classe (appelée le gestionnaire dans le code) contient toutes les collections de classes de stockage de données de la partie interne.

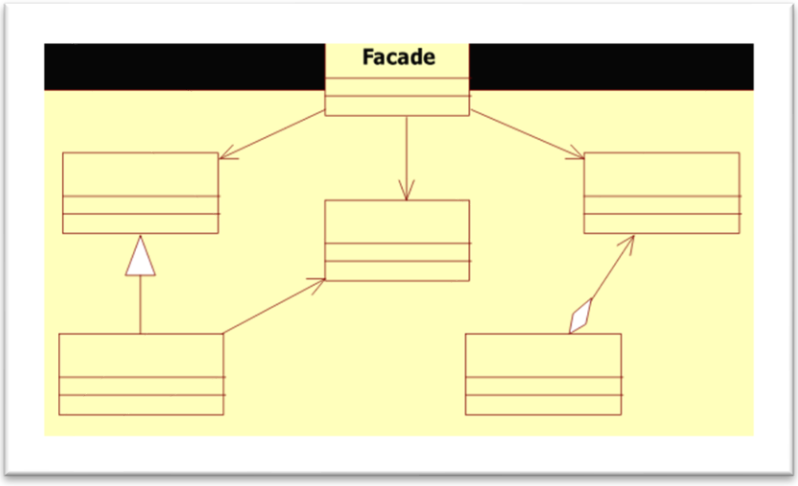


Diagramme d'une façade

Les classes de la partie externe n'ont aucun lien avec celles de la classe externe. Elles attendent une entrée et donnent une sortie. (À l'exception de la classe de paramètres settings). Elles ne nécessitent pas d'êtres instanciées. La seule classe qu'il faut instancier pour utiliser la bibliothèque est le gestionnaire.

Une classe d'importation a plusieurs stratégies (selon le type de fichier) et utilise une fabrique (factory) pour construire des classes de stockage de données à rendre au gestionnaire.

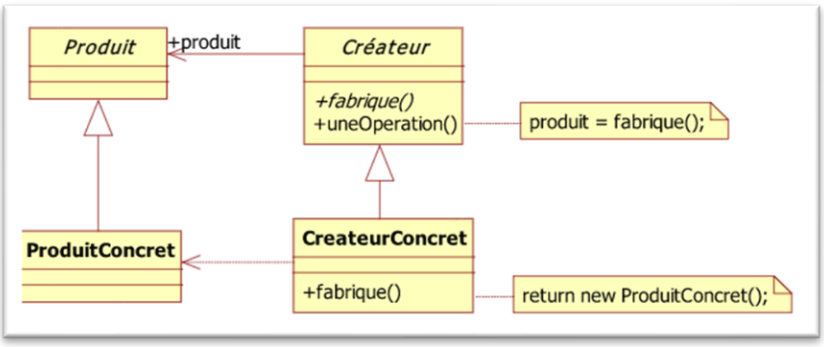


Diagramme d'une factory

Les classes de traitement sont surtout utilisées par les classes d'importation pour « traduire » du texte en données utilisables.

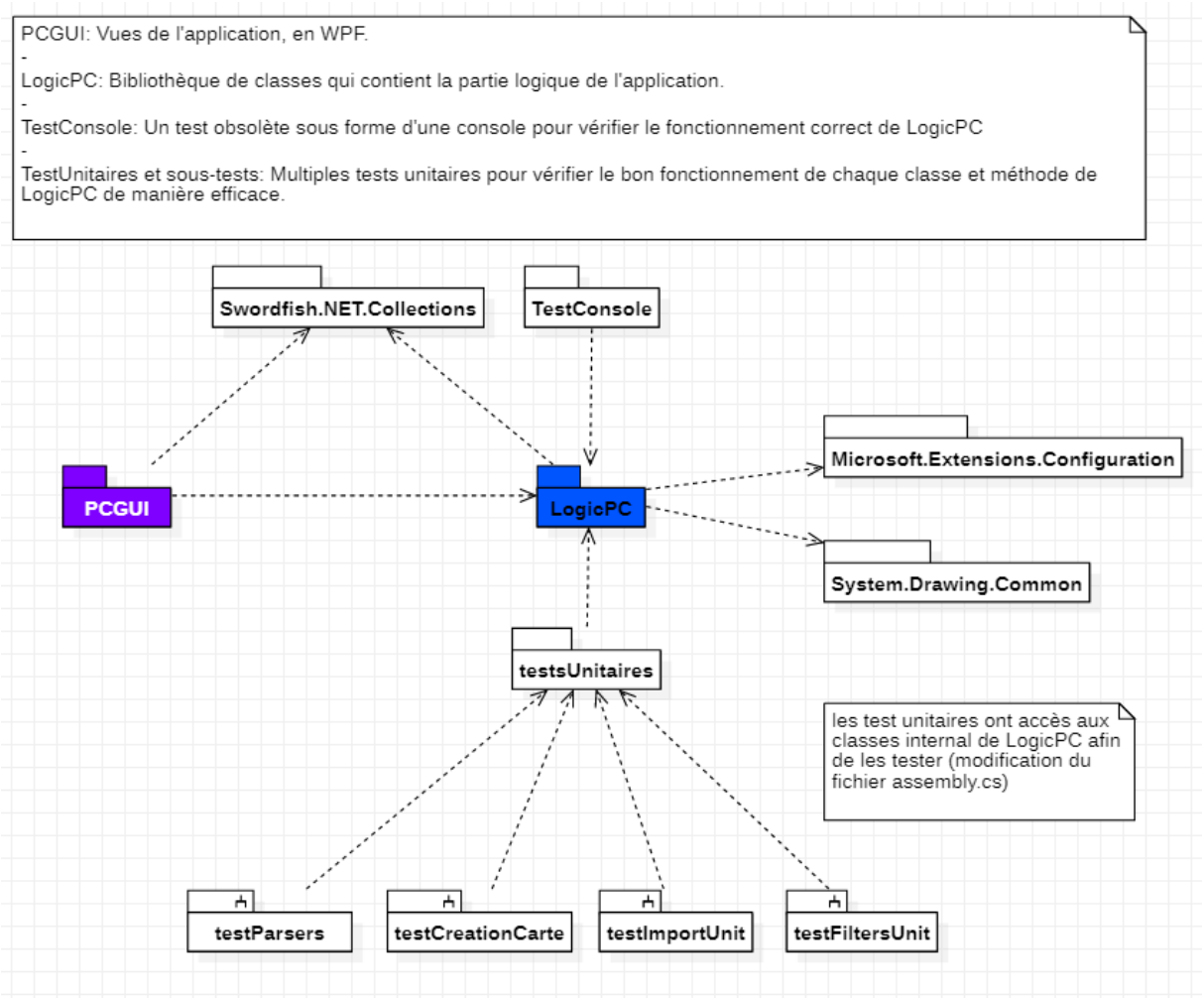
La classe d'import web est à part et sert à télécharger des ressources supplémentaires depuis internet (des images).

Les classes « externes » sont à part et sont surtout utilisées pour des recherches en leur donnant un dictionnaire et des filtres à appliquer.

La classe statique de paramètres est encore à part et contient de nombreuses valeurs utiles autant aux vues qu'au modèle (chemin de fichiers, délais, résolutions, etc…).

La version détaillée de ce diagramme et des patrons de conception utilisés est décrite dans la partie 3. Les codes couleurs pour les classes devraient être similaires pour plus de lisibilité.

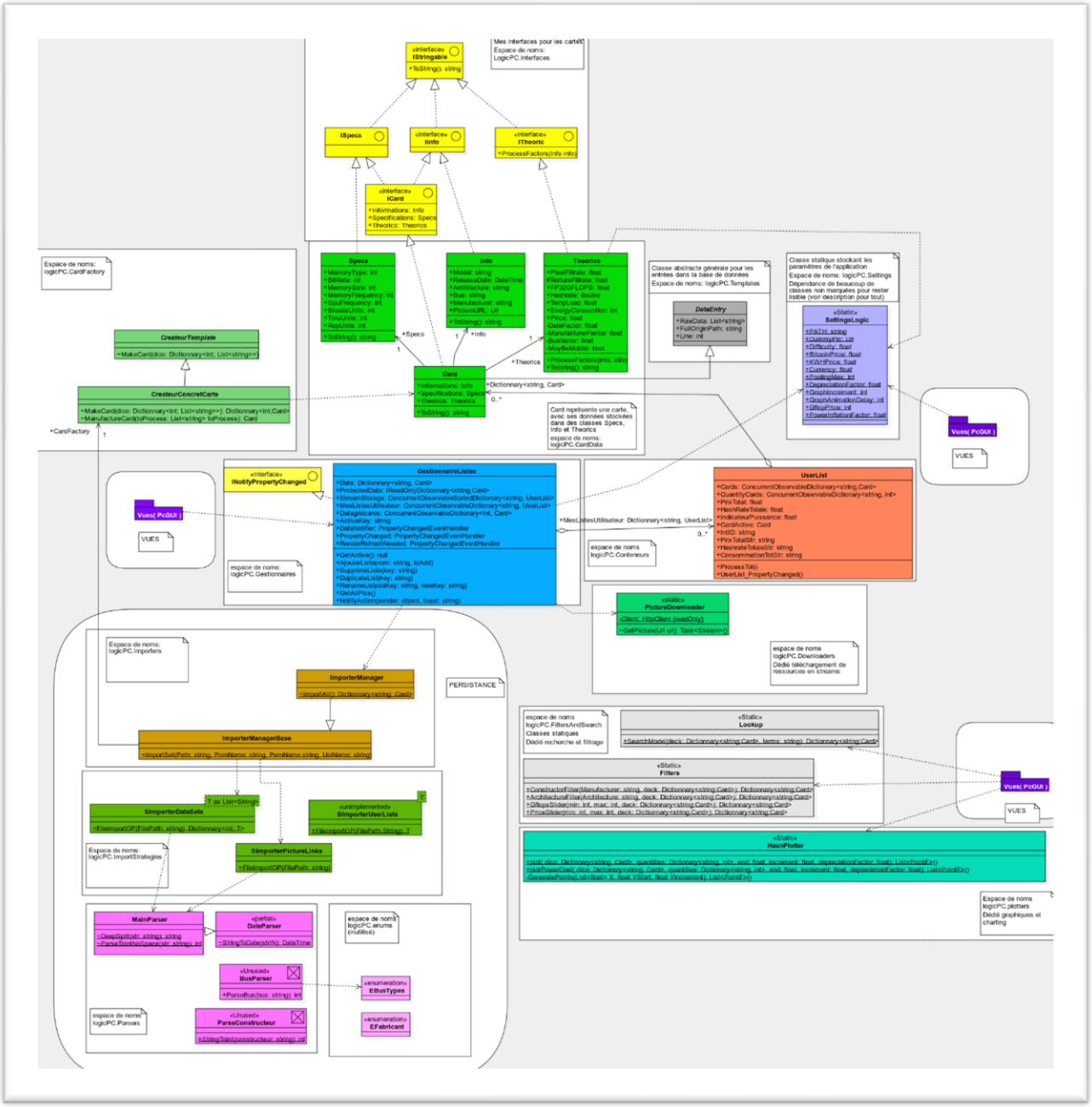
2) DIAGRAMME DE PAQUETAGE



Ce diagramme représente toutes les interdépendances du projet. A bien noter que seuls les packages enfants de testUnitaires ont un accès illimité aux classes et méthodes internal de LogicPC (édit fichier assemblyInfo.cs de `LogicPC/properties/`) cet édit évite de compromettre la fonctionnalité de la façade et de la bibliothèque en général juste pour les tests.

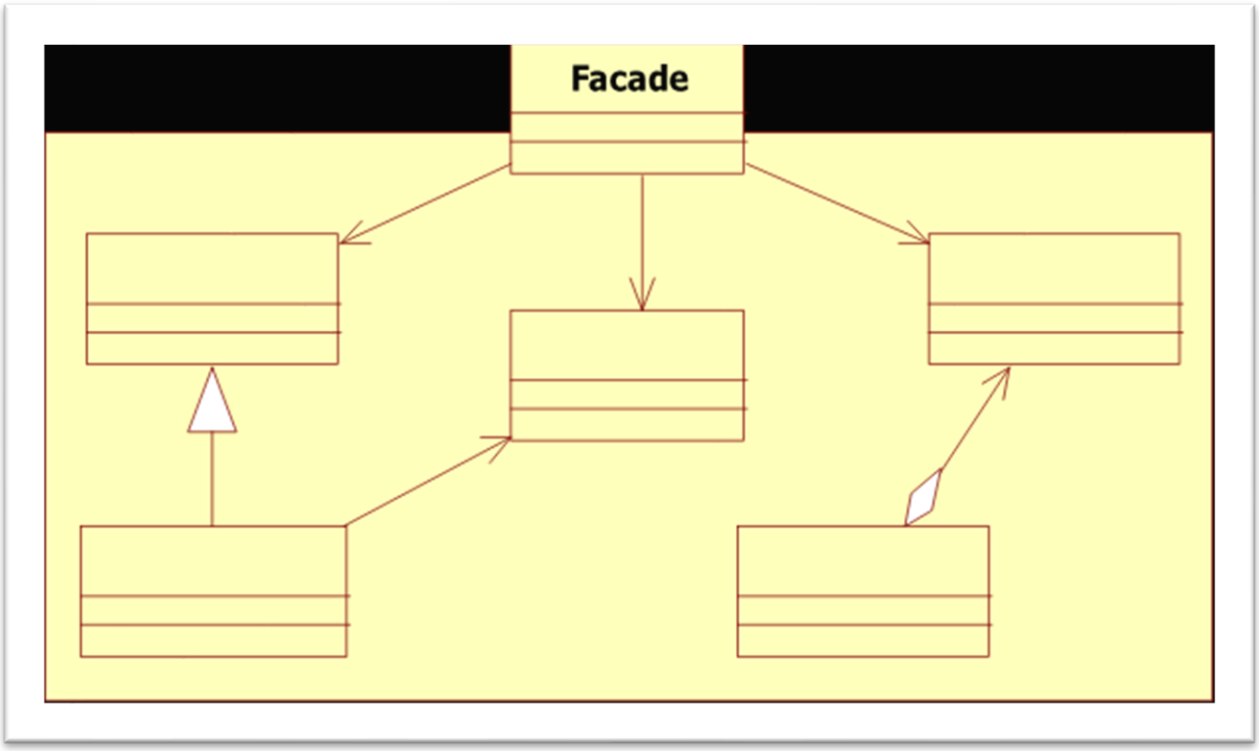
3) DESCRIPTION DU DIAGRAMME DE CLASSES

DIAGRAMME DE CLASSES COMPLET – BIBLIOTHEQUE PCLOGIC



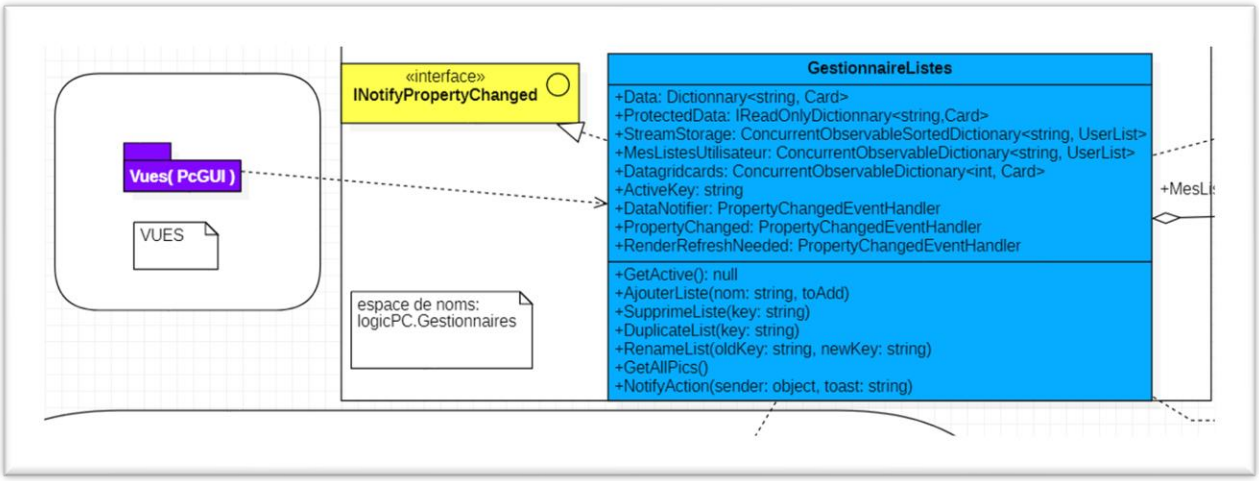
ESPACE DE NOMS LOGICPC.GESTIONNAIRES

Cet espace de noms contient la classe GestionnaireListes. Il s'agit en fait d'un manager général dans lequel sont stockées les données traitées de l'application. Cette classe agit comme une façade qui sera instanciée par les vues de l'application.



Ce patron de conception permet de grandement simplifier l'utilisation de la bibliothèque de classes en donnant aux vues une seule classe instanciable pour gérer toutes les données.

RESPONSABILITES DE LA CLASSE GESTIONNAIRELISTES

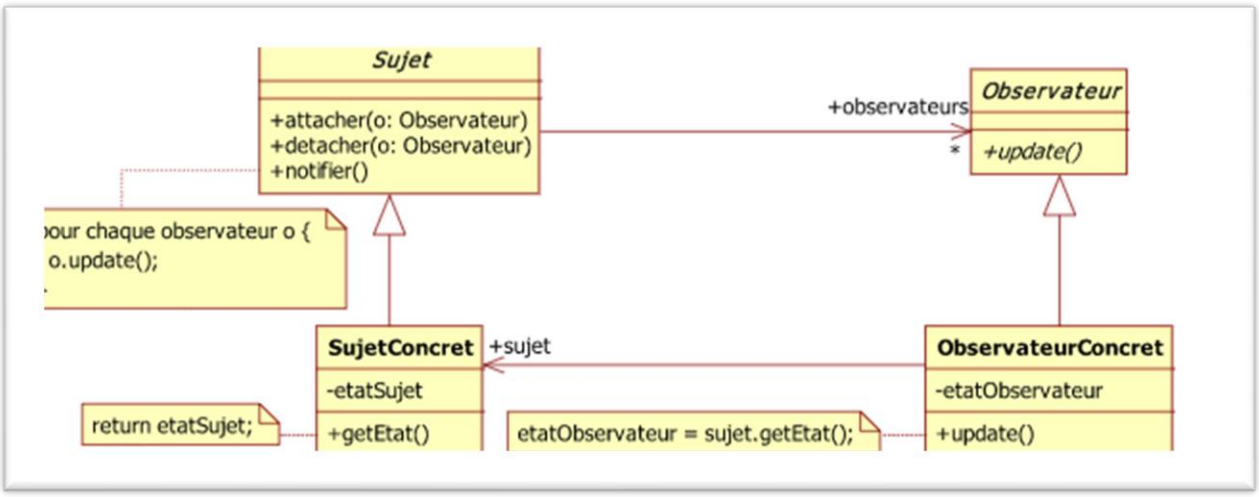


Cette classe a pour responsabilités :

- L'appel des méthodes statiques d'importation par GetAllPics(). La méthode est appelée dès l'instanciation de la classe GestionnaireListes.
- Le stockage direct de ces données importées dans le dictionnaire Data, puis, une fois le traitement terminé, l'enregistrement permanent de ces données dans le dictionnaire en lecture seule ProtectedData.
- Le stockage des listes utilisateur dans le dictionnaire MesListesUtilisateur.
- Le stockage de la clé de dictionnaire désignant la carte devant être affichée dans la vue par le Master Detail en tant que string dans ActiveKey.
- La gestion générale des listes utilisateur avec les méthodes AjouterListe(), SupprimeListe(), DuplicateList(), RenameList().
- Certains évènements auxquels d'autres classes peuvent s'abonner pour savoir si certaines valeurs ont changées.

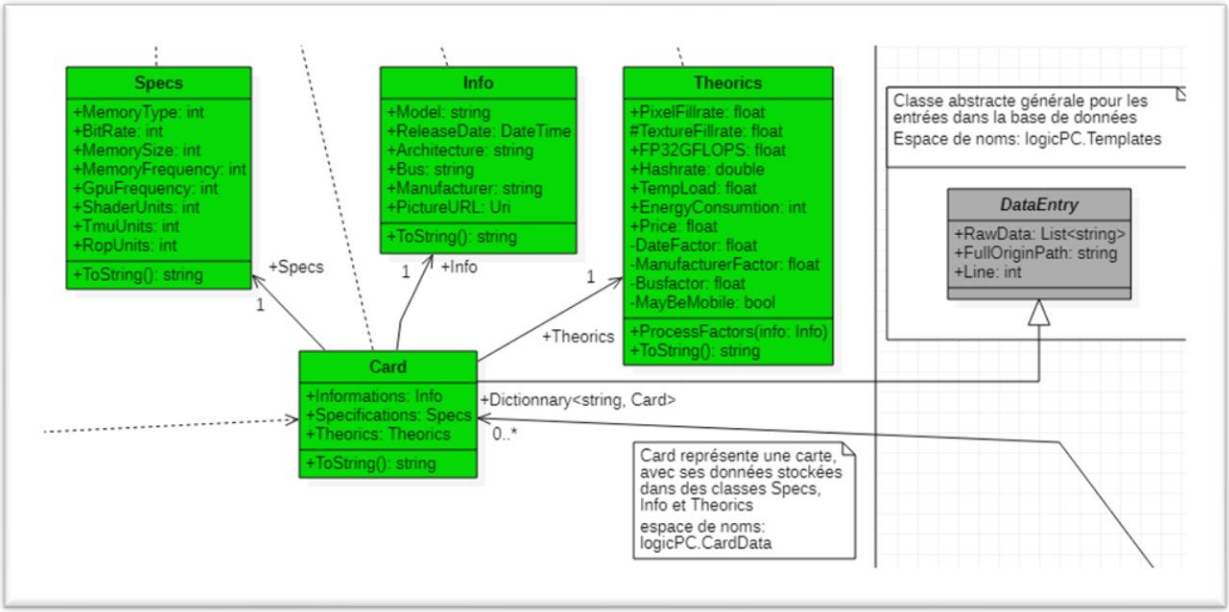
Cette classe possède également un champ PropertyChanged pour respecter le contrat de l'interface INotifyPropertyChanged. Une méthode non-référencée dans ce diagramme de classe y est attachée mais n'a aucune utilité autre que d'empêcher une exception où l'évènement PropertyChanged est lancé mais aucune méthode n'est abonnée à cet évènement. Le nom de cette méthode est `GestionnaireListes_PropertyChangedDummy()`.

Ce champ est un Observateur fourni par le langage C# directement :



Dans ce cas, le sujet est GestionnaireListes (MesListesUtilisateur plus spécifiquement), l'observateur est PropertyChanged. L'évènement est lancé dès qu'une des méthodes de gestion de listes se termine.

ESPACE DE NOMS LOGICPC.CARDDATA (ET LOGICPC.TEMPLATES)



La classe carte utilise un modèle (pas un patron de conception) composition-et-héritage pour contenir ses données. Elle dérive de la classe abstraite DataEntry qui est la classe la plus basique utilisable avec la bibliothèque de classes (juste les données brutes sous forme de strings, le chemin vers le fichier d'origine et le numéro de ligne d'où cette carte provient).

RESPONSABILITES DE LA CLASSE CARD

La classe Card instancie les classes Specs, Info et Theorics et dérive de *DataEntry*. Les classes de cet espace de noms, hormis Theorics, n'effectuent aucun traitement de données et ne servent qu'à stocker les données qui leur sont assignées lors de leur instanciation.

- Informations contient une classe Info qui sert à stocker les informations non-numériques sur la carte (nom, date de sortie, etc…).
- Specifications contient une classe Specs qui sert à stocker les informations numériques de la carte (surtout des indicateurs de performance).
- Les méthodes ToString() de Card et des classes qu'elle instancie retourne un string contenant les informations importantes de chaque classe.
- Theorics contient une classe Theorics qui ne contient – à l'origine – pas de données. Elle est instanciée après Informations et Specifications et son rôle est l'extrapolation de données à partir de celles contenues dans le dataset. Voir « Responsabilités de la classe Theorics ».

Les classes Info et Specs n'auront pas de description de responsabilité car elles sont assez simples et ne servent qu'au stockage d'informations.

RESPONSABILITES DE LA CLASSE THEORICS

Comme précité, la classe Theorics ne se voit pas assignée d'informations immédiatement. Elle va utiliser celles des classes Info et Specs pour construire les siennes.

- Les champs publics de Theorics contiennent les informations extrapolées à partir de celles d'Info et de Specs.
- Les champs privés de Theorics contiennent des « facteurs » qui sont utilisés dans le calcul des données de champ public.

[2]A terme, les données de cette classe seront largement utilisées pour pallier aux manque de données du dataset (comme le prix de la carte, ou sa réelle puissance). Bien que semi-fiable elle permet de donner des valeurs purement indicatives à l'utilisateur. Elles seront aussi utilisée pour créer des graphiques permettant de voir les relations puissance/consommation/temps/coût/revenu.

RESPONSABILITES DE LA CLASSE *DATAENTRY*

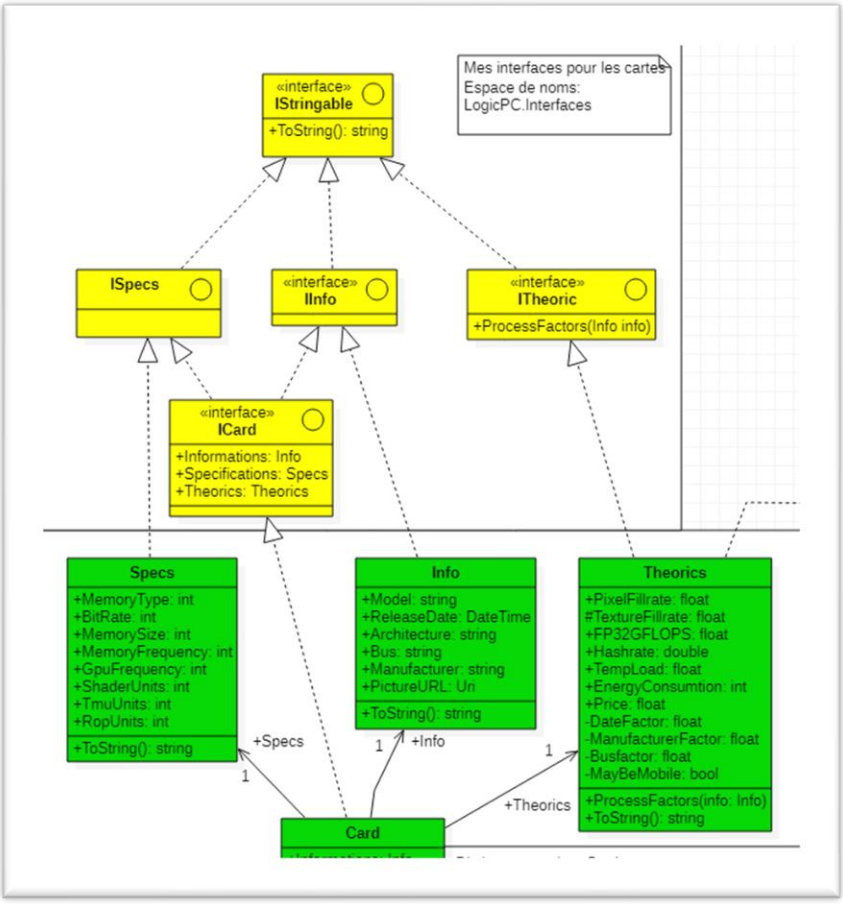
La classe abstraite DataEntry contient les données brutes récupérées par les classes d'importation de fichier. Le but réel de cette classe est double :

- Permettre la réutilisation facile de cette bibliothèque avec une autre classe dérivant de DataEntry que Card pour s'adapter à un autre type de dataset (une classe ASICS pour les professionnels par exemple).
- Permettre le réenregistrement de cette carte dans un autre fichier, sans avoir à passer toutes les données dans un parseur inverse (notez que les classes d'exportations qui utiliseraient un tel système ne sont pas encore implémentées).

Cette classe contient trois champs :

- RawData est une liste de strings non-deepsplit() [rop/tmu/etc intacts] qui contient les données brutes de la carte sans traitement (un simple join() suffit à retrouver le string original)
- FullOriginPath est un string qui indique le chemin complet vers le fichier d'origine de la carte (utile si la carte provient d'un dataset importé manuellement par l'utilisateur hors du PATH indiqué par Settings).
- Line est simplement la postion de cette carte dans le fichier d'origine (la ligne à laquelle elle a été lue).

ESPACE DE NOMS LOGICPC.INTERFACES



Il s'agit de l'espace de noms pour les interfaces de PcLogic. Les seules qui sont présentes pour le moment sont celles liées à la classe Card et aux classes qu'elle instancie. Les 3 seules interfaces intéressantes ici sont IStringable qui définit le contrat suivant :

Une classe doit avoir une méthode ToString() qui ne prend pas d'arguments et qui renvoie un string des informations de cette classe, autre que le ToString() d'objet par défaut.

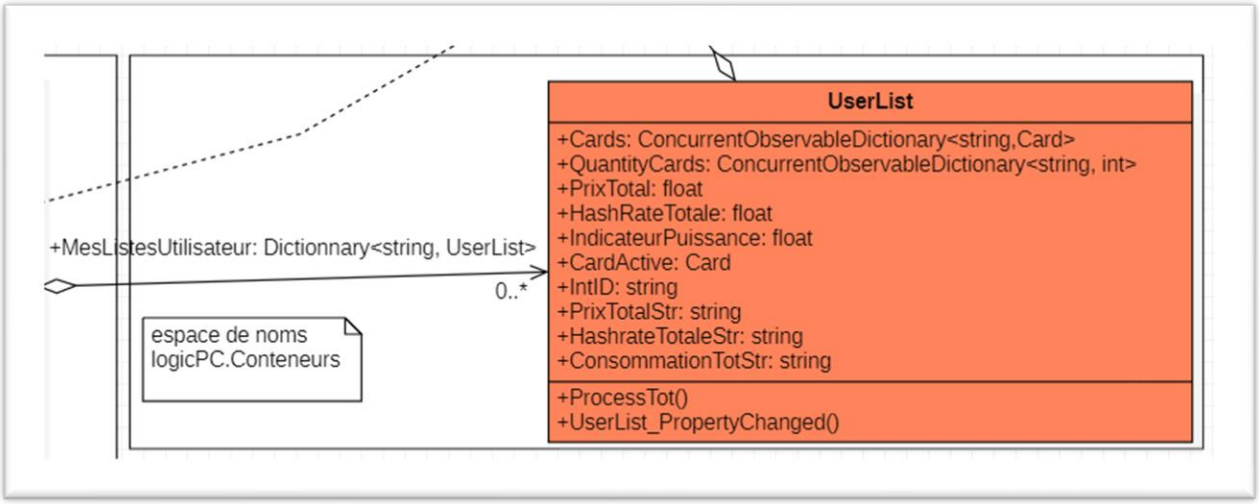
ITheoric :

Une classe doit avoir une méthode ProcessFactors() qui prend une classe info en argument

ICard :

Une classe doit avoir les propriétés Informations, Specficiations, et Theorics.

ESPACE DE NOMS LOGICPC.CONTENEURS

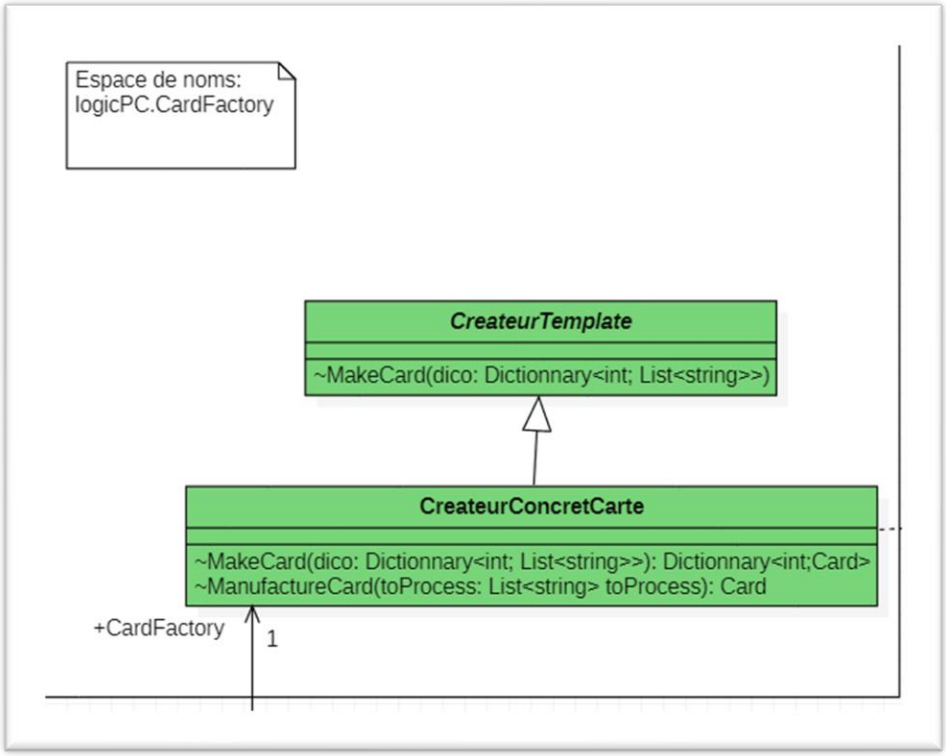


L'unique classe de cet espace de noms, UserList, est une classe de stockage de données. Elle contient une liste de cartes graphiques créé par l'utilisateur. Ses champs sont :

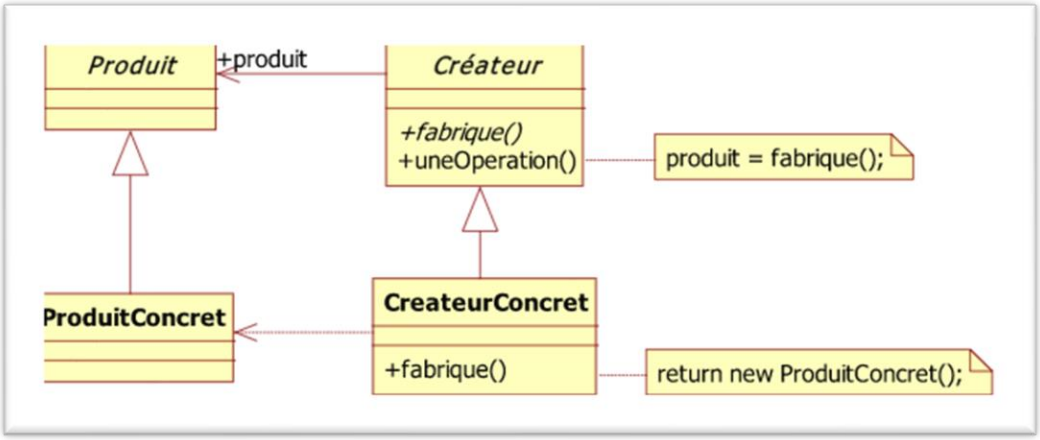
- Cards : Un dictionnaire de cartes qui contient toutes les cartes de la liste
- QuantityCards : Un dictionnaire constitué des mêmes clés que Cards, avec en valeur le nombre de cartes pour cette clé.
- PrixTotal : Le coût direct total pour l'achat de toutes les cartes de cette liste.
- HashRateTotale : la hashrate totale de toutes les cartes de la liste.
- IndicateurPuissance : le total des FP32GFLOPS de toutes les cartes de la liste.
- CardActive : la carte active de la liste (actuellement présentée)
- IntID : La clé identifiante de cette UserList dans le dictionnaire parent (sera utilisé pour l'exportation)

Toutes les méthodes de gestion liées à cette classe sont dans le GestionnaireListes.

ESPACE DE NOMS LOGICPC.CARDFACTORY



Comme son nom l'indique, cet espace de noms contient une Factory :



Ici le créateur est la classe abstraite createurTemplate, Produit est DataEntry, CreateurConcret est CreateurConcretCarte et ProduitConcret est Card.

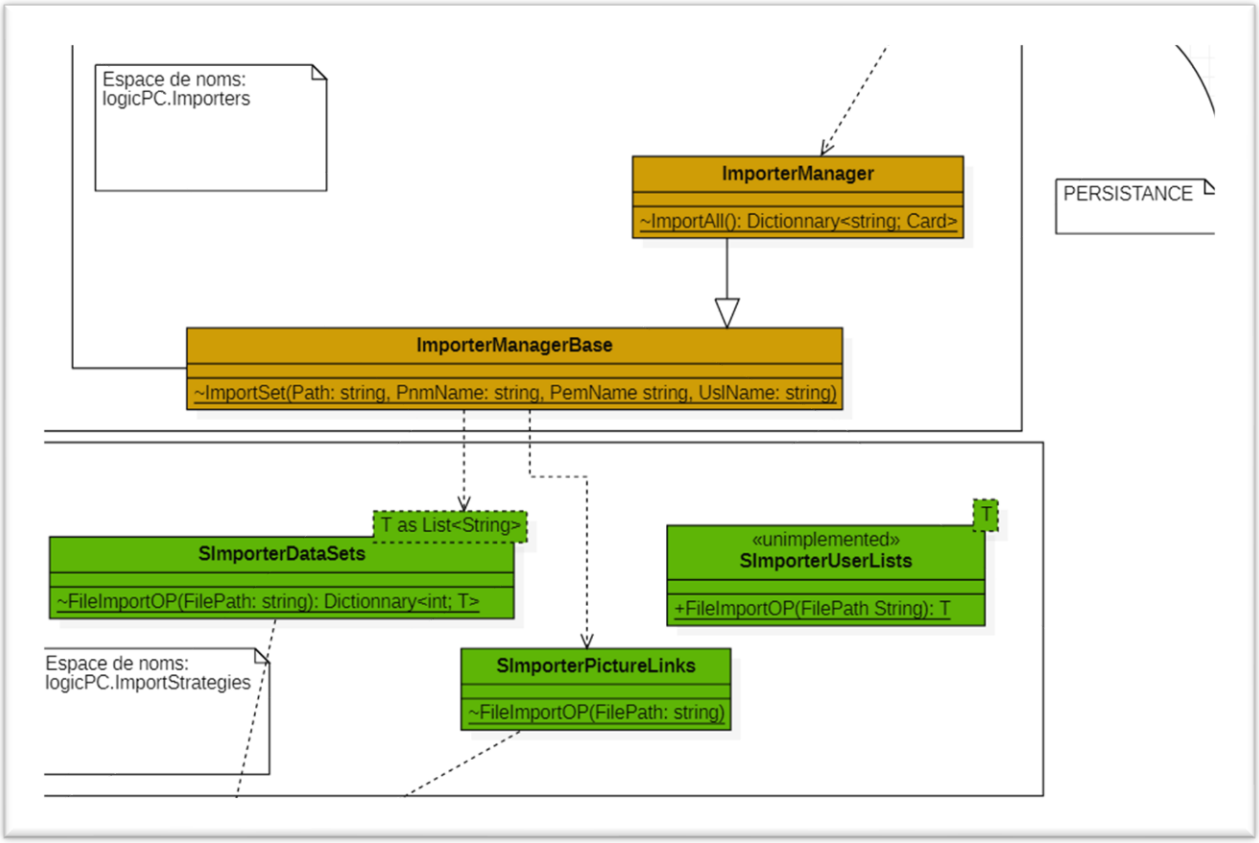
La particularité de cette Factory est qu'elle ne renvoie pas qu'une seule Card en prenant un seul string en argument. Elle prend un dictionnaire de listes de strings, et renvoie un dictionnaire de cartes.

Cette classe est instanciée par ImporterManagerBase dans l'espace de noms logicPC.importers.

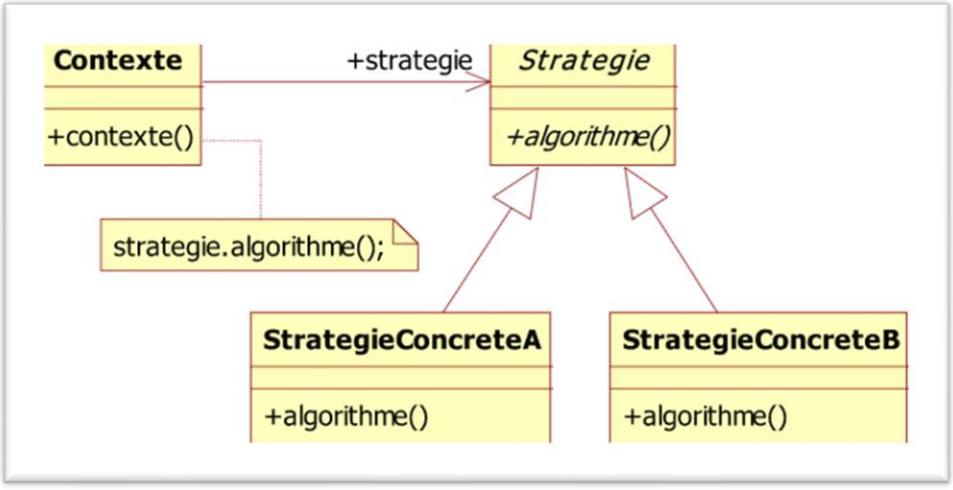
ANNEXES – CLASSES DE PERSISTANCE

Certaines classes liées à la persistance étaient requises pour tester correctement LogicPC, comme elles ne devraient pas être dans le même projet que la logique de l'application, je vais les décrire rapidement ici :

LOGICPC.IMPORTERS/LOGICPC.IMPORTSTRATEGIES

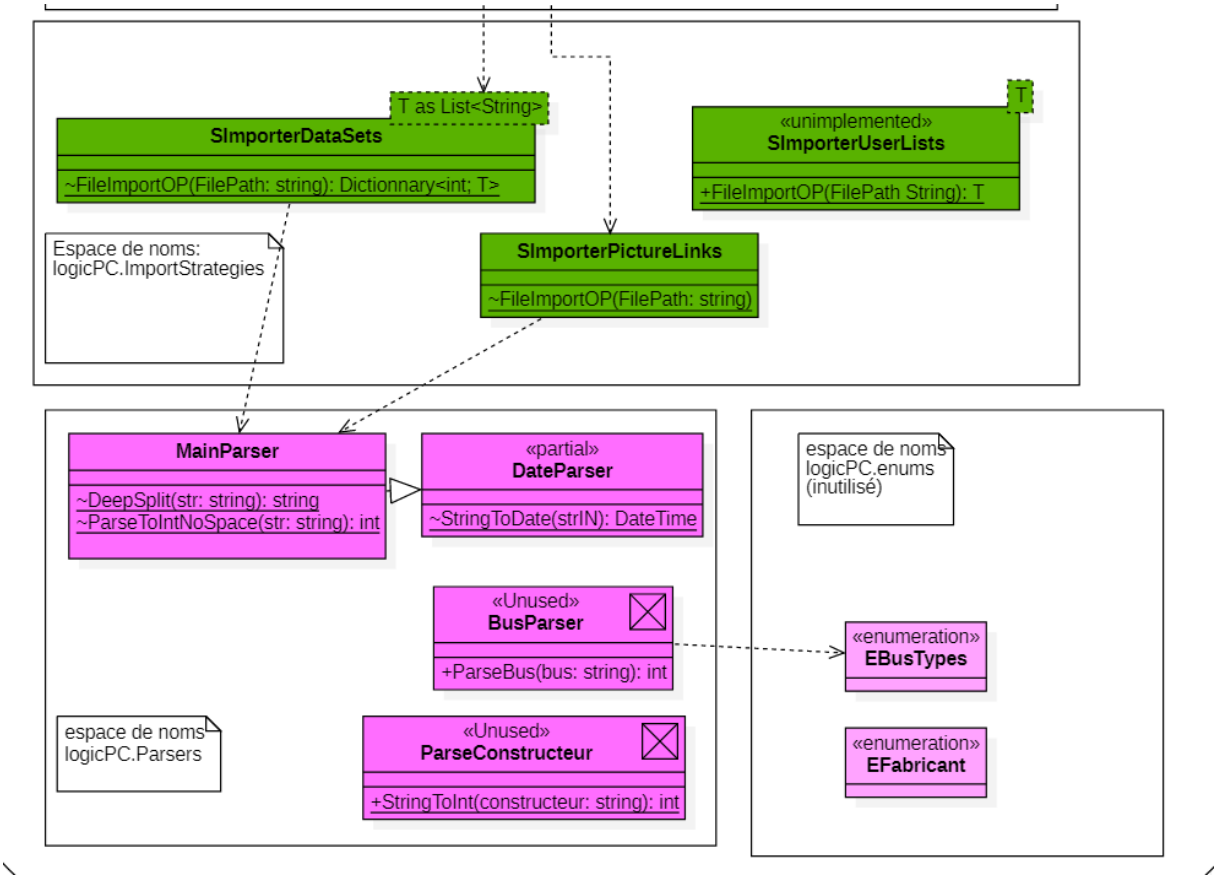


A l'origine les classes de cet espace de noms suivaient un patron de conception de stratégies (d'où leur nom) :



Cela a été abandonné car les stratégies auraient été trop différentes les unes des autres.

Pour faire court, les classes de ces deux espaces servent à importer un ou plusieurs couples de fichier .pnm (Card data) et .pem (Card Picture) provenant du chemin précisé dans logicPC.Settings. Elles traitent ces fichiers en utilisant les classes de l'espace de nom logicPC.Parsers.

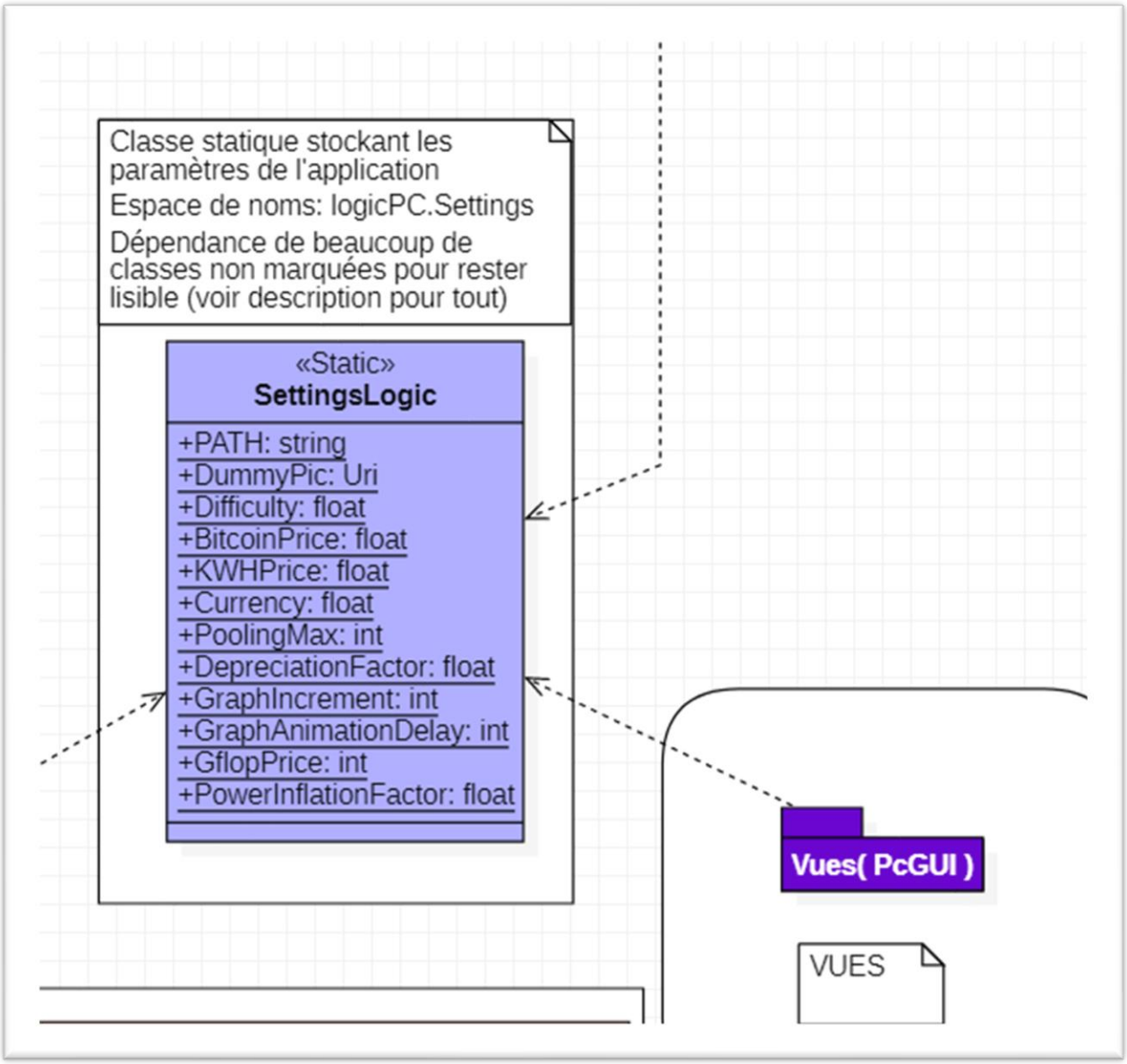


ImporterManagerBase instancie Une CardFactory. Une fois le processus d'importation et de traitement terminé, un dictionnaire<string, Card> est rendu à la classe apellante (ici GestionnaireListes).

Seules les parties indispensables de ces classes ont été programmées et elles seront en grande partie restructurées lors de leur transfert au projet dédié persistance.

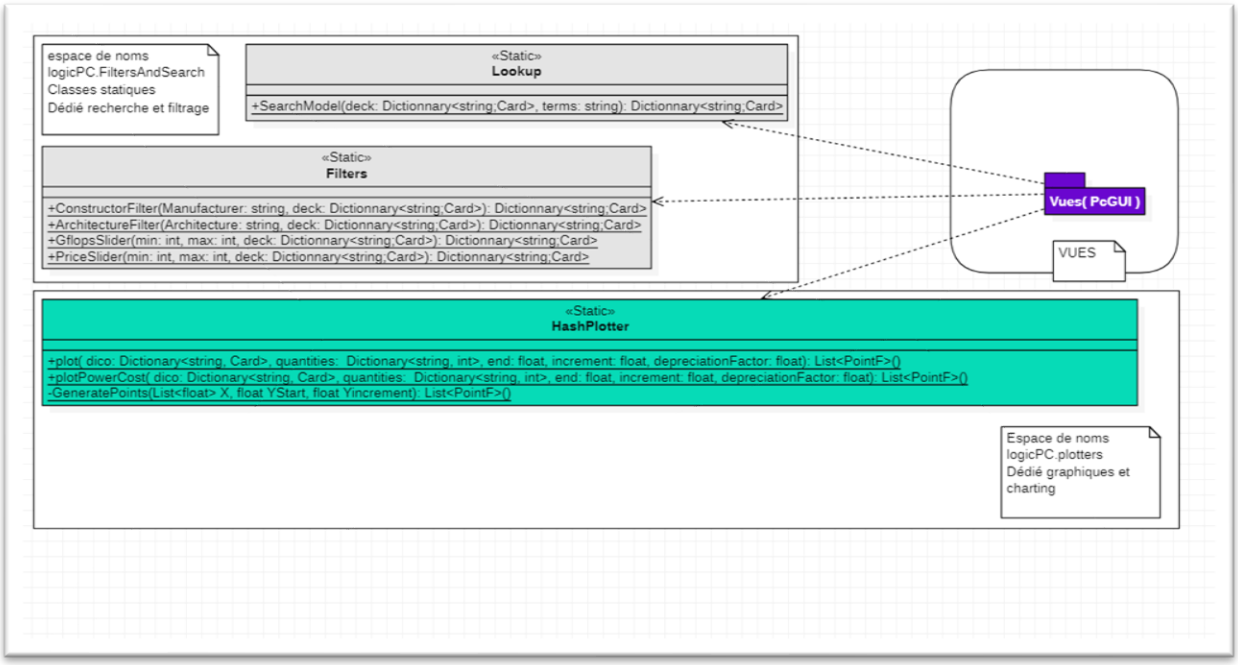
ANNEXE 2 – CLASSES STATIQUES INDEPENDANTES

LOGICPC.SETTINGS



Comme son nom l'indique, cet espace de noms contient une classe statique qui sert à stocker les paramètres de l'application.

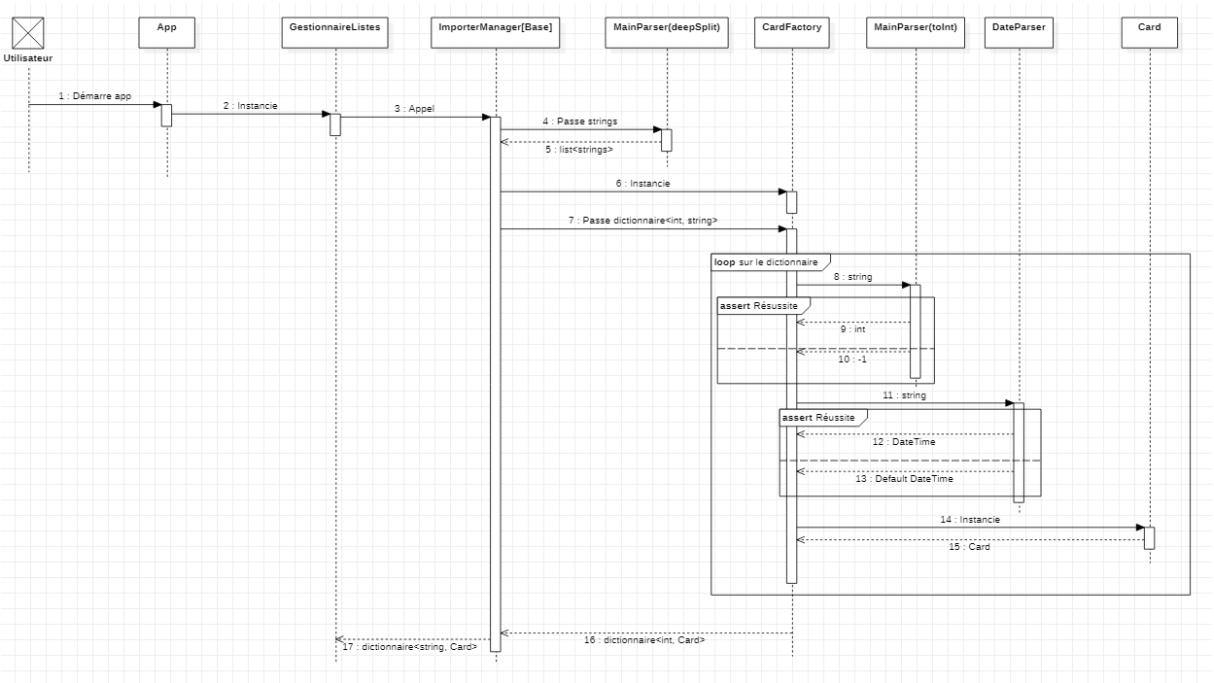
ANNEXE 3 – CLASSES STATIQUES « EXTERNES »



Ces 3 classes sont utilisées par les vues pour la recherche, le filtrage et les graphiques. Elles attendent toutes les valeurs dont elles ont besoin en entrée et ne requièrent pas d'informations depuis le reste du modèle.

4) DIAGRAMMES DE SEQUENCE

Import de données :



Ajout d'une liste via le gestionnaire :