

Documentation De PcParted

L'almanach du mineur de cryptomonnaies.

GEOFFRE YORICK

LIEN VERS LA VIDEO DE PRESENTATION RAPIDE : [HTTPS://YOUTU.BE/2ZD3-R0DSNK](https://youtu.be/2ZD3-R0DSNK)

GITHUB: [HTTPS://GITHUB.COM/KANKEN6174/PCPARTED](https://github.com/KANKEN6174/PCPARTED)

TABLE DES MATIERES

Documentation IHM	2
Contexte de l'application : PcParted	2
Personas	2
User stories : Bill Turner	3
User stories : Jean Durot	3
Sketchs/Storyboards (wireframe)	4
Diagramme de Flux	4
Diagramme de cas d'utilisation	4
CAS « Choisir une base de données »	5
Cas « Lister les GPUs dans la base de données »	6
Cas « Accéder aux détails d'un gpu »	7
Cas « Gérer ses listes »	7
CAS « Modifier les paramètres de l'application »	8
Cas « Projeter les revenus et déficits d'une liste de GPU sur graphique »	8
Diagramme de classes complet – Bibliothèque PcLogic	13
Espace de noms logicPC.Gestionnaires	13
Responsabilités de la classe GestionnaireListes	14
Espace de noms logicPC.CardData (et logicPC.Templates)	16
Responsabilités de la classe Card	17
Responsabilités de la classe Theorics	17
Responsabilités de la classe <i>DataEntry</i>	18
Espace de noms logicPC.Interfaces	18
Espace de noms logicPC.Conteneurs	19
Espace de noms logicPC.CardFactory	20
ANNEXES 1 – Classes de persistance	22
logicPC.Importers/logicPC.ImportStrategies	22
Espace de noms persistance	23
Persistance du cache	24
ANNEXE 2 – Classes statiques indépendantes	25
LogicPC.Settings	25
ANNEXE 3 – Classes statiques « externes »	25

DOCUMENTATION IHM

CONTEXTE DE L'APPLICATION : PCPARTED

Je souhaite créer une application qui me permettrait de facilement calculer le revenu que pourraient générer les diverses cartes graphiques que j'ai amassées au cours du temps. Simplement se dire qu'une carte plus récente sera meilleure pour du minage n'est pas forcément vrai et cela me semble être un sujet intéressant.

Dans cette application, il sera d'obtenir des informations utiles sur des cartes graphiques dans le cadre de minage de cryptomonnaie. PcParted dispose de nombreux filtres pour affiner votre recherche et trouver la carte qui vous correspond.

Vous pourrez ainsi, entre-autres, choisir d'afficher les cartes d'un certain constructeur, d'une certaine série ou de les trier par hashrate (anglicisme signifiant la capacité d'une unité de calcul à « miner » une cryptomonnaie, donné en mégahash par secondes). Chaque carte qui semble répondre à vos besoins peut être ajoutée à une liste à droite de l'écran en la glissant ou en appuyant sur le bouton « ajouter » (mode détaillé).

Pour chaque carte affichée il est possible d'obtenir plus de détails en cliquant dessus. Cela lance alors le mode détaillé qui contient toutes les informations techniques dont vous pourriez avoir besoin.

Même si à la base cette application sera dédiée à des cartes graphiques, son mode de fonctionnement sera assez flexible pour permettre d'intégrer tout ce que l'on veut à sa liste. Tant qu'il existe un moyen d'extrapoler une puissance de hachage théorique de l'appareil, ou que l'entrée customisée dispose d'un fichier attaché précisant prix, consommation, et hashrate, l'application saura traiter ces données.

Un mode de prédiction graphique sera disponible, pour mieux visualiser les chiffres et aider dans la prise de décision sur une certaine période.

Enfin, chaque carte ajoutée à la liste de droite sera comptabilisée dans le prix, la consommation et la hashrate totale (avec notation en cas d'incompatibilité ou de blocage de minage par le constructeur).

PERSONAS



Bill Turner

"je veux une analyse détaillée,
des chiffres et des résultats!"

propriétaire d'une ferme bitcoin

Age : 39

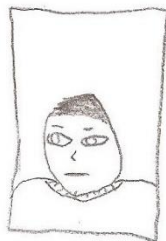
profession : cloud services

Situation : célibataire

Revenus : 106 000 /an

Profil tech : bonne
connaissance des sujets
liés à la cryptomonnaie
et à l'informatique

veut : optimiser une ferme dont il
dispose déjà.



Jean Durot

"Je cherche la simplicité, rien
de trop compliqué..."

Age : 43

profession : avocat

Situation : marié

Revenus : 81 300 /an

Profil tech : Sait utiliser
un pc de manière assez
avancée mais ça
s'arrête là.

veut : se lancer dans le minage

USER STORIES : BILL TURNER

En tant que responsable des services cloud dans une grande entreprise, je recherche une application qui me donne des informations utiles et détaillées, avec une option pour conserver une ou plusieurs listes de mineurs afin de faire correspondre les données de l'application avec les mineurs dont je dispose déjà. Etant déjà bien versé dans l'art du minage, les données ne me feront pas peur.

Une inclusion des cartes dédiées spécifiquement au minage (AntMiners) serait une addition appréciable.

USER STORIES : JEAN DUROT

En tant que débutant dans le milieu du minage de cryptomonnaies et dans le milieu informatique, je veux une interface simple et intuitive, afin de rendre mon expérience plus facile. J'ai déjà essayé des outils de l'industrie plus complexes et ils m'ont rapidement découragé. Je n'ai pas besoin de grands détails mais d'une mise en évidence de ce qui est important.

Je veux juste me lancer dans quelque chose de nouveau, et le monde de la crypto me paraît si mystérieux...

SKETCHS/STORYBOARDS (WIREFRAME)



wire.bmpr

DIAGRAMME DE FLUX

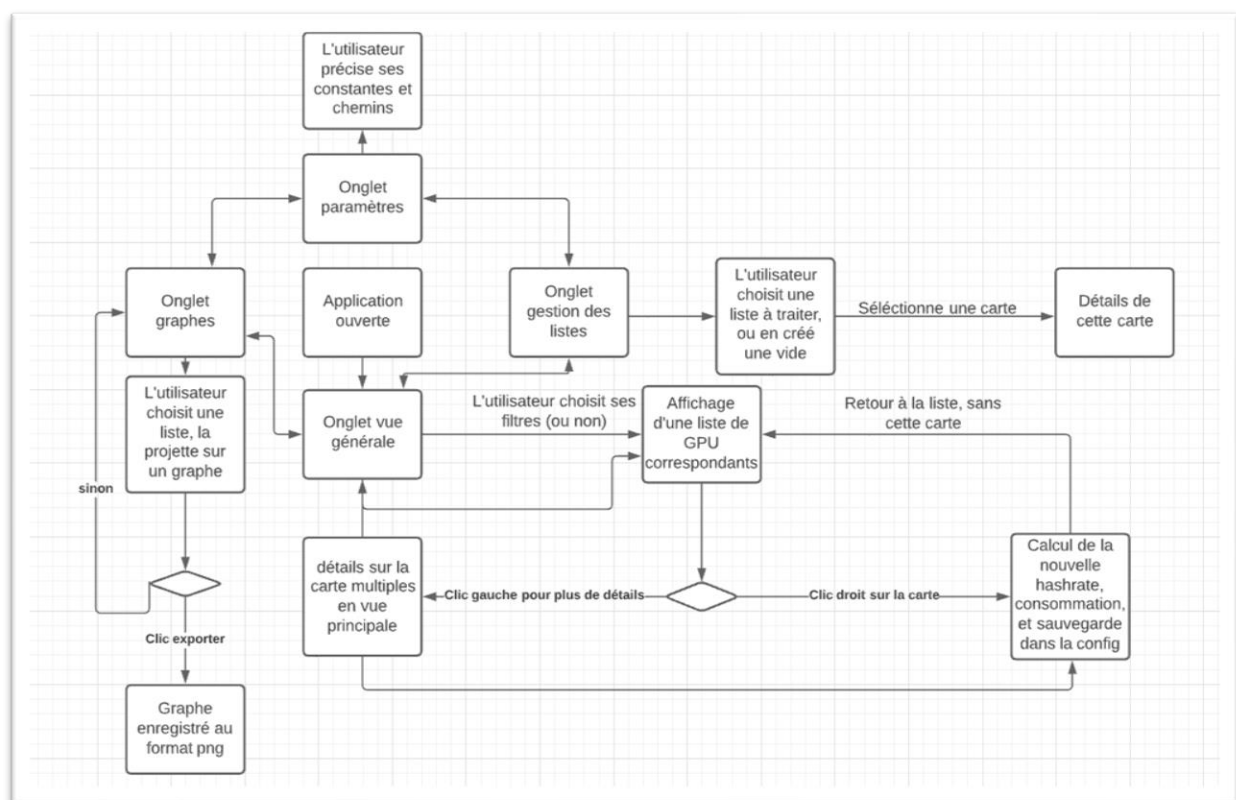
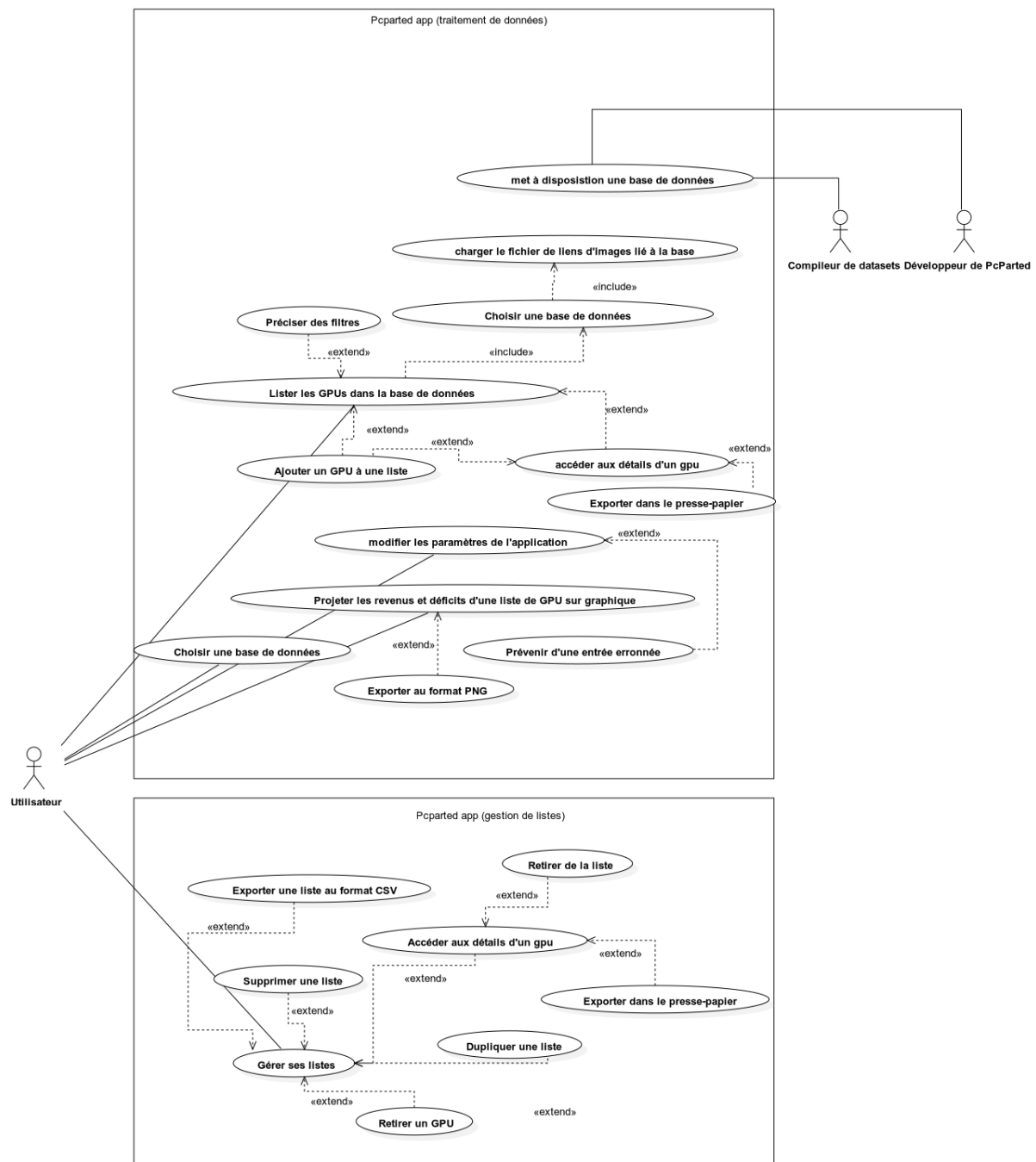



DIAGRAMME DE CAS D'UTILISATION



(voir UseCaseDiagram1.png ou svg)

CAS « CHOISIR UNE BASE DE DONNEES »

Nom	Choisir une base de données
Objectif	Choisir une BDD de cartes graphiques à charger dans l'application
Acteurs principaux	Utilisateur
Acteurs secondaires	∅

Conditions initiales	-L'utilisateur dispose d'un fichier de Basxe De Données (BDD) compatible avec l'application (.pnm)
Scénario d'utilisation	<p>-L'utilisateur précise un chemin dans la boîte de dialogue ou parcourt son disque vers le fichier pnm.</p> <p>-Le système vérifie l'intégrité du fichier (si corrompu ou incorrect, conditions de fin 2)</p> <p>-Le système charge le fichier .pem (fichier de liens d'images) indiqué par le fichier .pnm (si absent, conditions de fin 3)</p> <p>-Le système charge le fichier .pwr (fichier de données de consommation, de prix et de hashrate) (si absent, conditions de fin 4).</p> <p>-Le système affiche les données chargées (conditions de fin 1)</p>
Conditions de fin	<p>1) Fichiers pnm, pem et pwr corrects et intègres : la BDD est chargée dans l'application.</p> <p>2) Fichier .pnm corrompu, introuvable ou ne peut pas être ouvert. Le système indique un code erreur et ne charge pas la base de données.</p> <p>3) Fichier .pem manquant ou incorrect (nombre de lignes différent), la nouvelle base de données sera chargée mais avec des images d'espace réservé [placeholders basés sur la marque de la carte].</p> <p>4) Fichier .pwr manquant ou incorrect (nombre de lignes différent), la nouvelle base de données sera chargée mais les données telles que le prix, la hashrate et la consommation seront extrapolées mathématiquement en fonction de la marque, de la date de sortie,</p> <div style="text-align: right;">  </div> <p>du nombre de cœurs, de la fréquence, ect... [un symbole sera présent en haut à gauche des cartes extrapolées avec une tooltip expliquant sa signification].</p>

CAS « LISTER LES GPUS DANS LA BASE DE DONNEES »

Nom	Lister les GPUs dans la base de données
Objectif	Lister les gpus présents dans la base de données actuellement chargée
Acteurs principaux	Utilisateur
Acteurs secondaires	∅

Conditions initiales	-Au moins une liste existe avec au moins un GPU
Scénario d'utilisation	<p>-L'utilisateur ouvre l'onglet « vue générale » ou lance l'application (action par défaut au lancement de l'application) (conditions de fin 1)</p> <p>-L'utilisateur précise des filtres et / ou effectue une recherche (conditions de fin 2)</p>
Conditions de fin	<p>1) Tous les GPUs de la base de données choisie sont affichés, triés par hashrate par défaut. [Priorité aux non-extrapolés de ~10%]</p> <p>2) Les GPUs remplissant les conditions des filtres et/ou de la recherche</p>

	sont affichés, triés de la même façon que dans le cas 1)
--	--

CAS « ACCEDER AUX DETAILS D'UN GPU »

Nom	Accéder aux détails d'un gpu
Objectif	Obtenir plus d'informations sur un GPU en particulier, [et] l'ajouter à la liste courante
Acteurs principaux	Utilisateur
Acteurs secondaires	Ø

Conditions initiales	-Au moins une liste existe avec au moins un GPU -Les GPUs ont été listés comme dans le cas précédent
Scénario d'utilisation	-L'utilisateur clique [clic gauche] sur le GPU qui l'intéresse, cela ouvre une version plus grande de la carte avec une description plus complexe contenant les spécifications de cette carte graphique OU l'utilisateur effectue un [clic droit] sur la carte du GPU, ce qui l'ajoute immédiatement à la liste (conditions de fin 1) -L'utilisateur clique sur le bouton « ajouter » dans la carte agrandie (conditions de fin 1) -L'utilisateur clique sur le bouton « retour » dans la carte agrandie (conditions de fin 2)
Conditions de fin	1) Le GPU sélectionné est ajouté à la liste courante 2) Retour à la vue précédente, la carte reprend son aspect normal

CAS « GERER SES LISTES »

Nom	Gérer ses listes
Objectif	Dupliquer, modifier, supprimer, exporter une liste existante ou créer une liste vide.
Acteurs principaux	Utilisateur
Acteurs secondaires	Ø

Conditions initiales	-Au moins une liste existe -L'application a les droits d'écriture dans son dossier courant
Scénario d'utilisation	-L'utilisateur sélectionne une liste à traiter dans la comboBox dédiée -Il peut cliquer sur un des 5 boutons à droite de la comboBox pour respectivement Créer une nouvelle liste vide (conditions de fin 3), dupliquer, modifier, ou supprimer la liste (conditions de fin 1). Il est également possible de l'exporter au format .csv (conditions de fin 4) -Lorsqu'une liste est choisie, ses GPUs s'affichent dans la dataGrid en-dessous des boutons. En cliquant sur un GPU dans la liste, les détails de ce

	GPU s'affichent. Il est alors possible de le retirer de la liste (conditions de fin 2) ou de l'exporter dans le presse-papier (conditions de fin 5)
Conditions de fin	<ol style="list-style-type: none"> 1) Les modifications sont apportées à la liste sélectionnée 2) Le GPU sélectionné est supprimé de la liste 3) Une nouvelle liste vide est créée 4) Un fichier nom_de_la_liste.csv est créé dans le dossier ./PcParted/exported/csv/ ou dans le dossier spécifié dans les paramètres 5) Les données du GPU sélectionné sont enregistrées dans le presse-papier (données du fichier .pnm + données du fichier .pem [si présent] + données du fichier .pwr [si présent])

CAS « MODIFIER LES PARAMETRES DE L'APPLICATION »

Nom	Modifier les paramètres de l'application
Objectif	Changer des paramètres divers pour le calcul et la sauvegarde de données
Acteurs principaux	Utilisateur
Acteurs secondaires	Ø

Conditions initiales	-L'application a les droits d'écriture dans son dossier courant
Scénario d'utilisation	<p>-L'utilisateur précise des constantes qui serviront aux calculs de l'application, ainsi que d'autres comme le chemin de sauvegarde et de lecture par défaut des bases de données et de listes (à la convenance de l'utilisateur)</p> <p>-L'utilisateur appuie sur le bouton « appliquer ». Une série de vérifications intégrées se lance. Les champs invalides seront entourés en rouge par le système avec un message aidant à la correction. Tant qu'ils n'auront pas été corrigés, le bouton appliquer sera désactivé.</p>
Conditions de fin	<ol style="list-style-type: none"> 1) Le fichier .ini de l'application enregistre les changements et l'application continue son fonctionnement normal

CAS « PROJETER LES REVENUS ET DEFICITS D'UNE LISTE DE GPU SUR GRAPHIQUE »

Nom	Projeter les revenus et déficits d'une liste de GPU sur graphique
Objectif	Prédire les revenus et dépenses sur une durée T indiquée
Acteurs principaux	Utilisateur
Acteurs secondaires	Ø

Conditions initiales	-Au moins une liste existe avec au moins un gpu à l'intérieur.
Scénario d'utilisation	<p>-L'utilisateur sélectionne l'onglet « graphe » de l'application.</p> <p>-Il précise une liste de GPUs à utiliser, et donne la durée souhaitée en jours, mois ou années.(si aucune liste disponible, conditions de fin 2) (si durée</p>

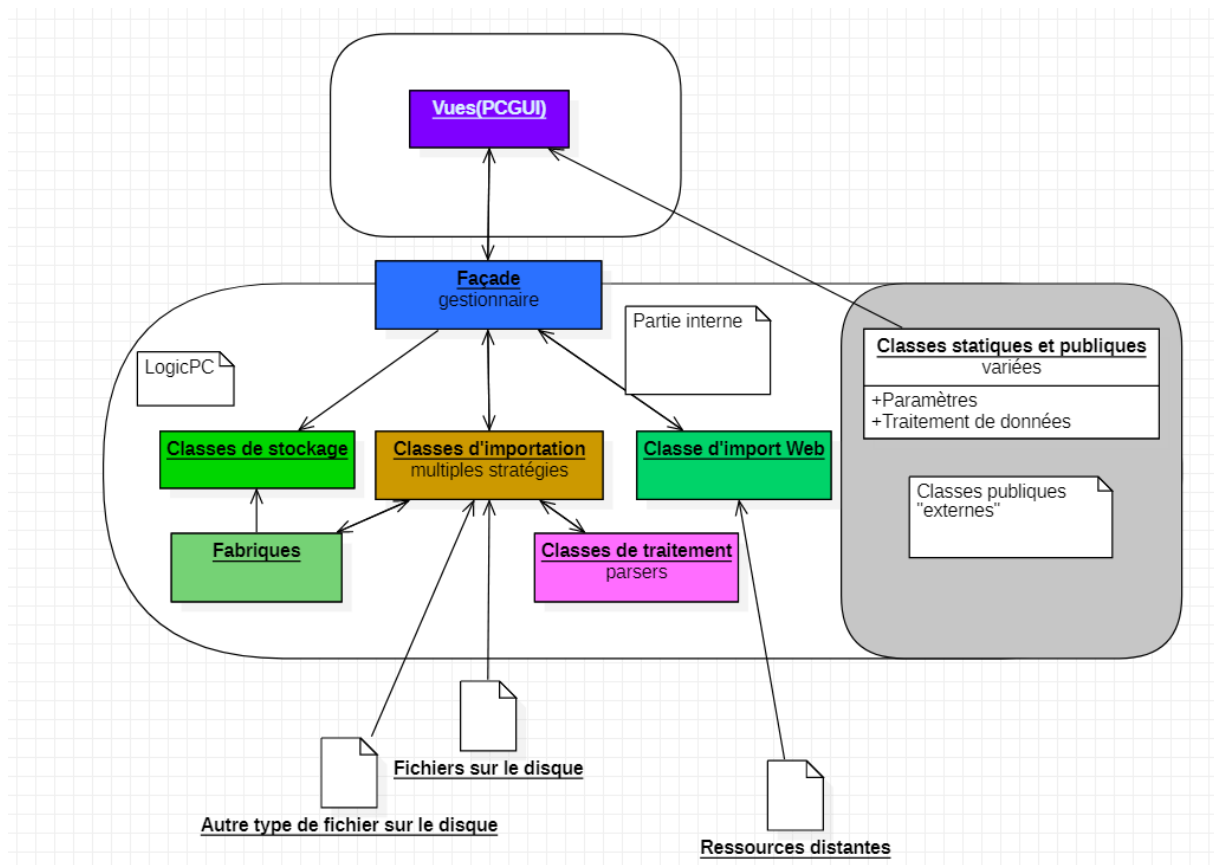
	incorrecte, conditions de fin 3) -L'utilisateur -L'application génère le graphique (conditions de fin 1)
Conditions de fin	<ol style="list-style-type: none"> 1) Le graphe est généré et visible dans l'application, il peut également être exporté en tant qu'image. 2) Sans liste disponible, le bouton « générer » restera désactivé avec une indications « aucune liste disponible ». 3) La textBox de durée sera encadrée en rouge et le bouton « générer » sera indisponible

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.

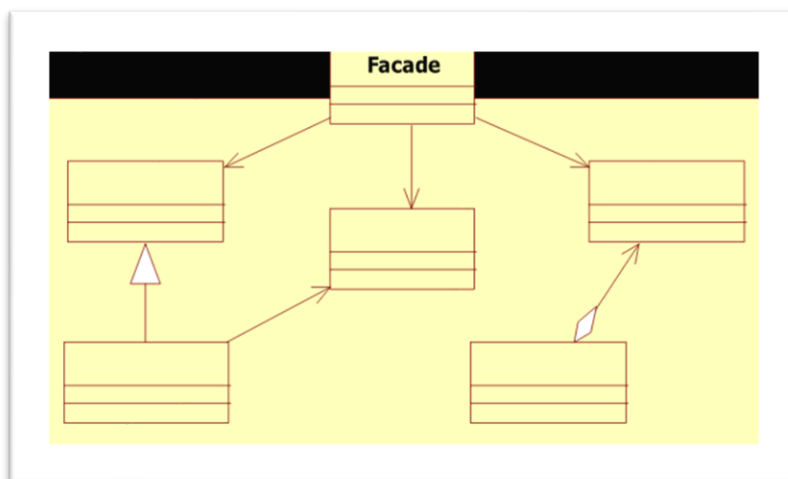


Diagramme d'une façade

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'être 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.

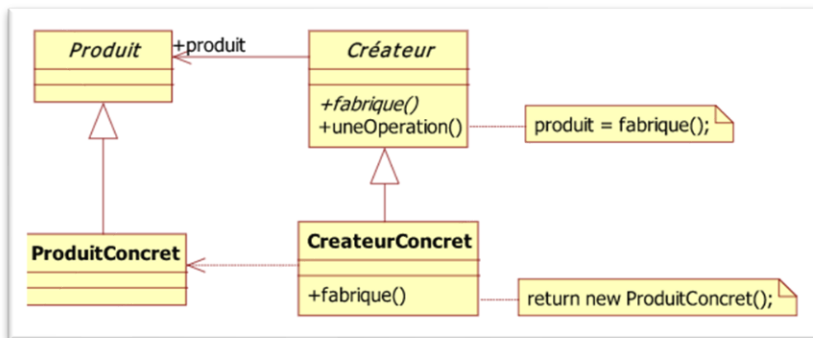


Diagramme d'une factory

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.

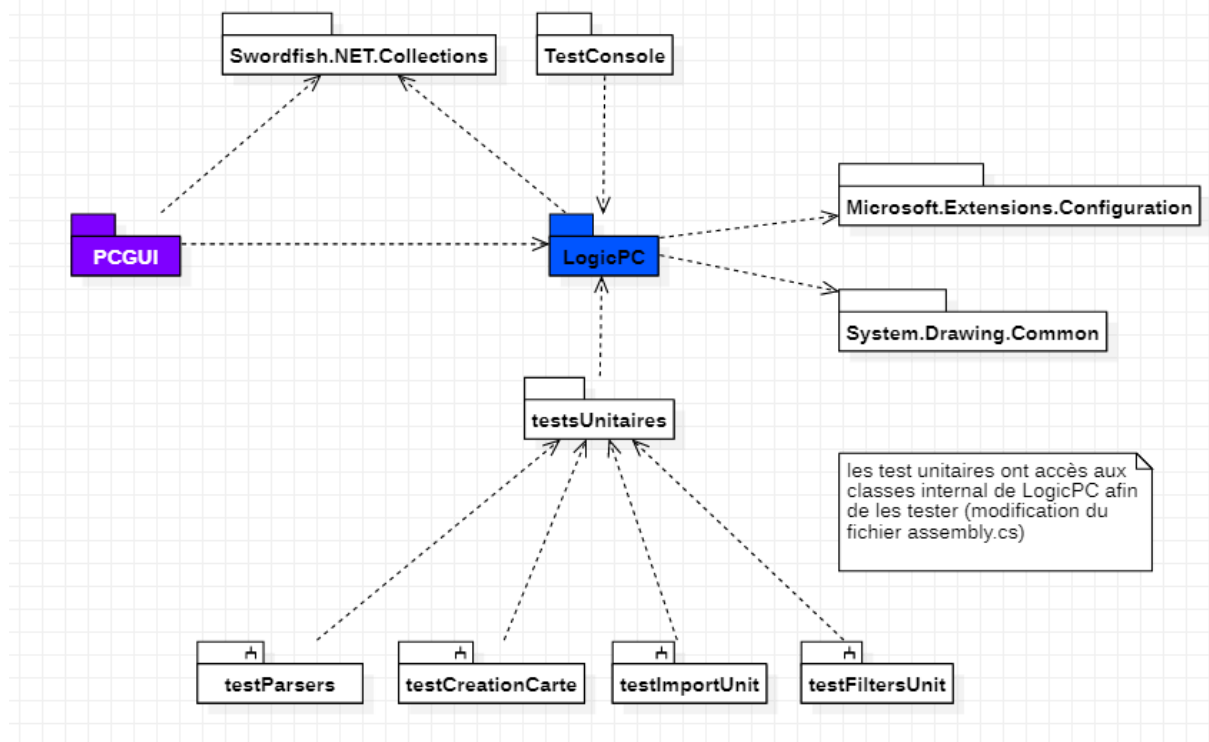
2) DIAGRAMME DE PAQUETAGE

PCGUI: Vues de l'application, en WPF.

- LogicPC: Bibliothèque de classes qui contient la partie logique de l'application.

- TestConsole: Un test obsolète sous forme d'une console pour vérifier le fonctionnement correct de LogicPC

- TestUnitaires et sous-tests: Multiples tests unitaires pour vérifier le bon fonctionnement de chaque classe et méthode de LogicPC de manière efficace.

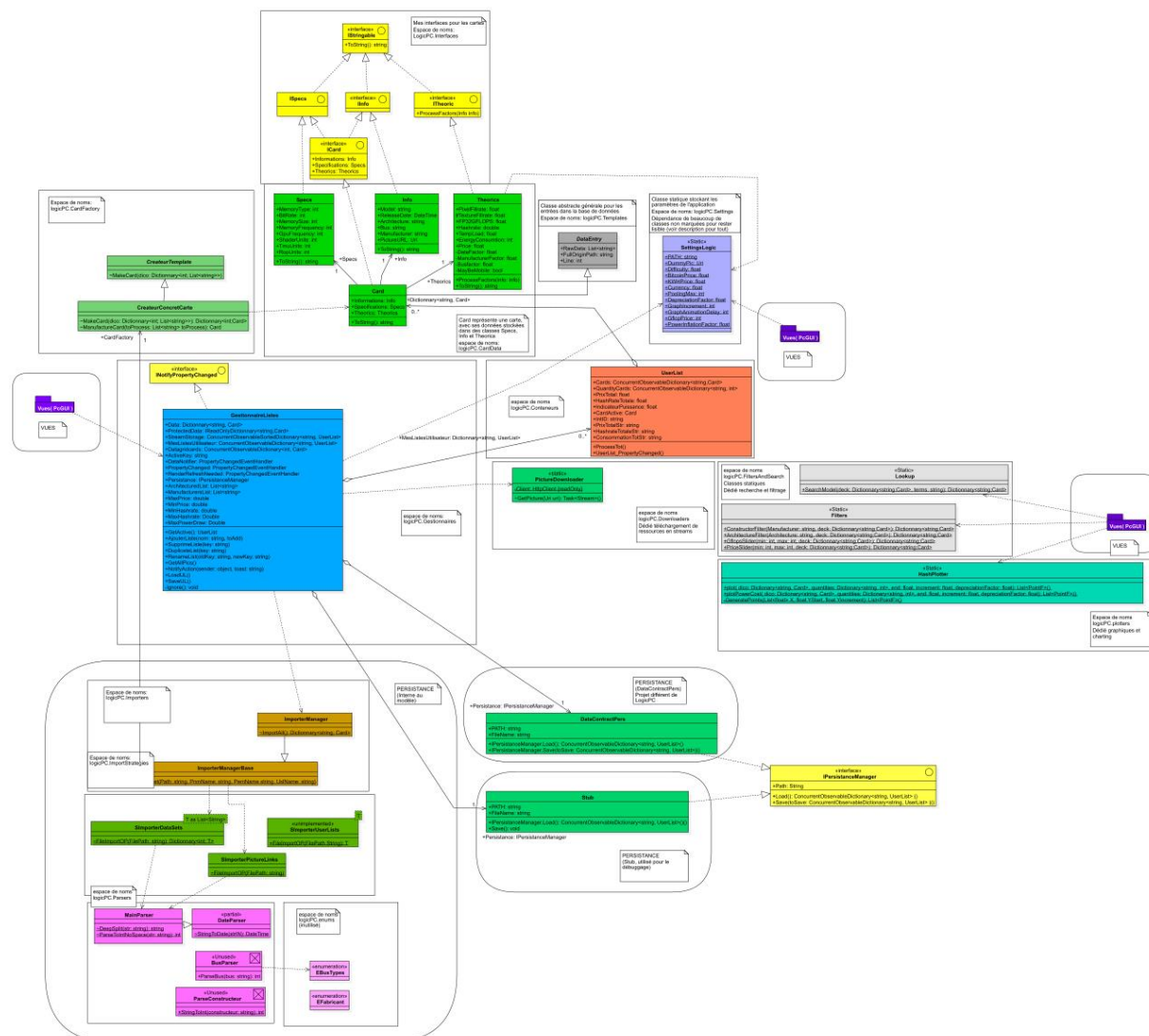


Ce diagramme représente toutes les interdépendances du projet. A bien noter que seuls les packages enfants de testUnitaires ont un accès illimité aux classes et méthodes internal de LogicPC (édit fichier assemblyInfo.cs de LogicPC/properties/) cet édit évite de compromettre la fonctionnalité de la façade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

Swordfish est le package Nuget d'origine des dictionnaires ConcurrentObservableDictionary utilisés à travers logicPC.

3) DESCRIPTION DU DIAGRAMME DE CLASSES

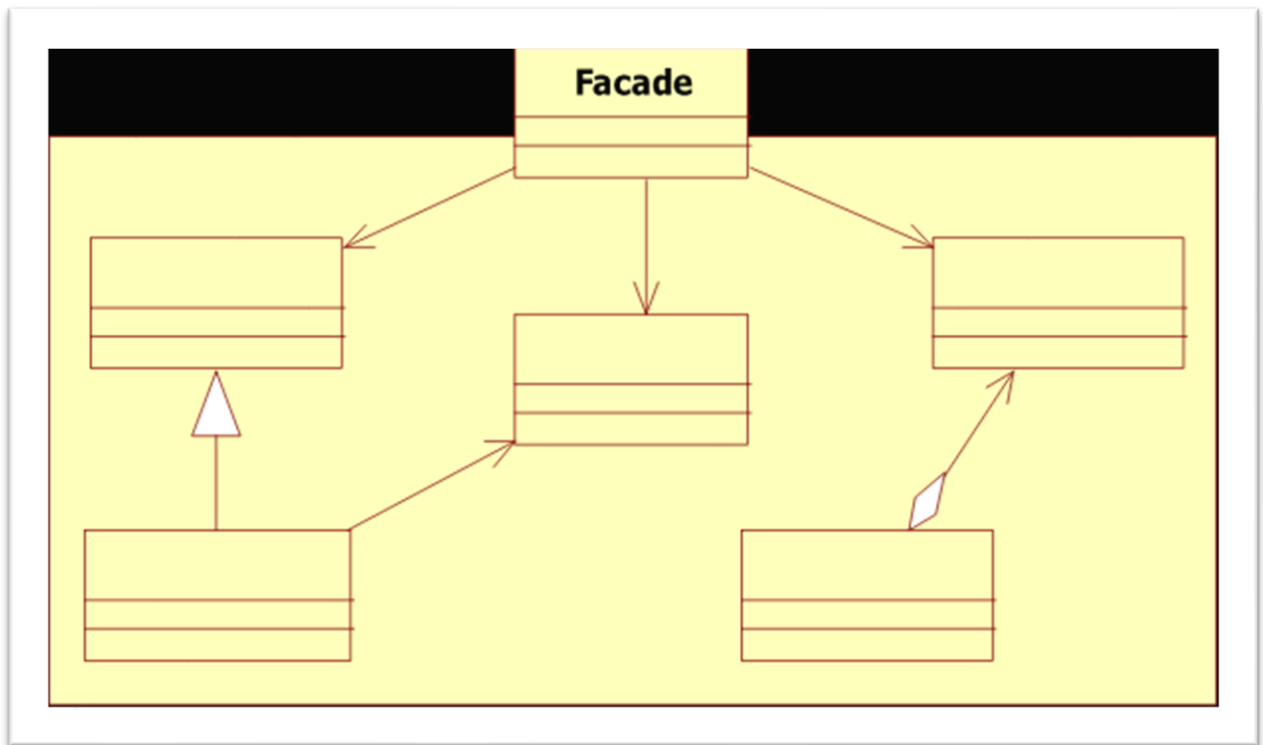
DIAGRAMME DE CLASSES COMPLET – BIBLIOTHEQUE PCLOGIC



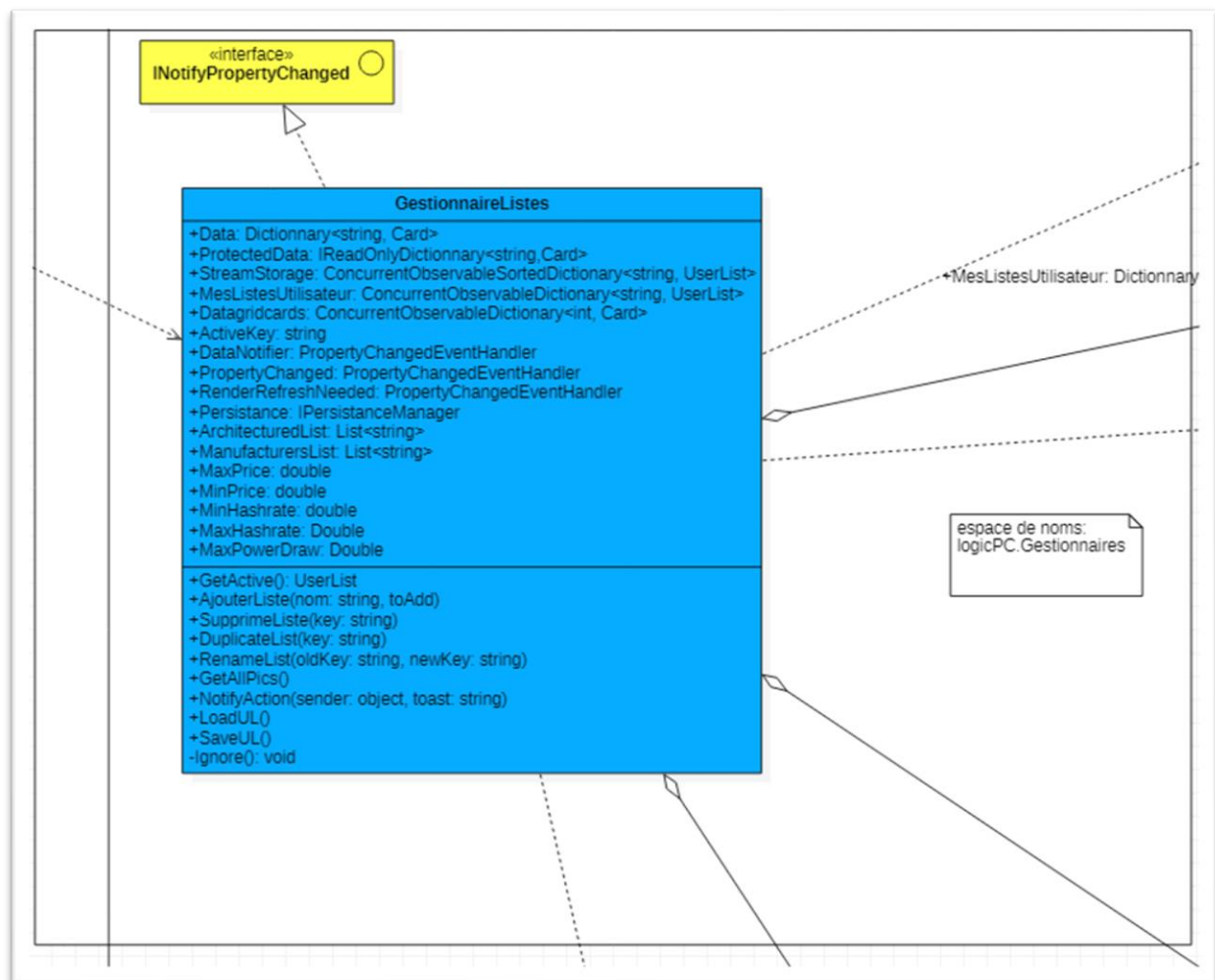
Le diagramme de classes est trop grand pour être visualisé ici correctement, veuillez vous référer au .svg ou au fichier StarUML pour une meilleure compréhension.

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.

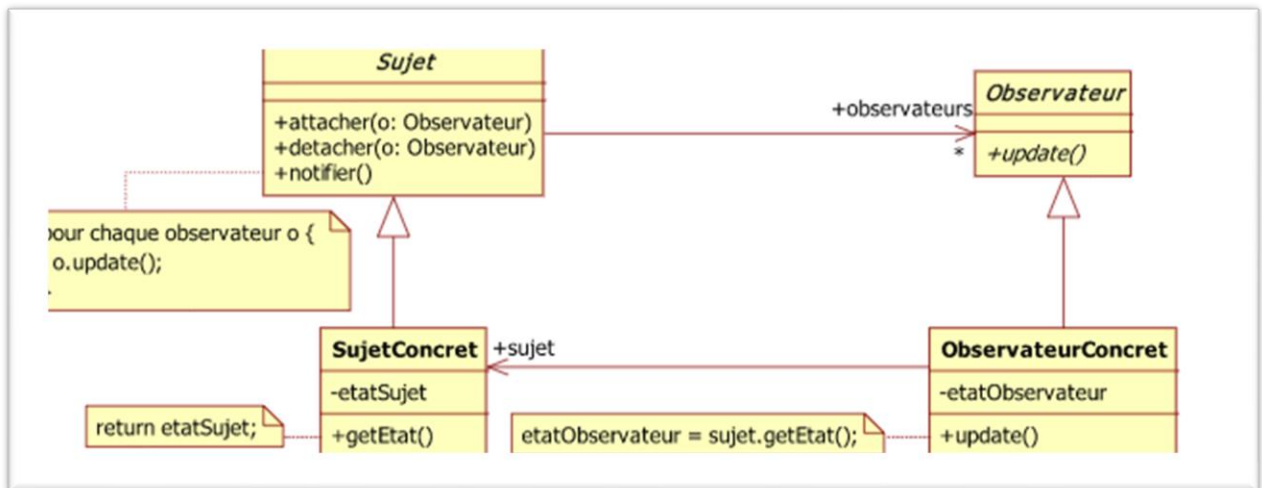


Cette classe a pour responsabilités :

- 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.
- Elle contient les extrêmes du dataset pour aider le travail des filtres
- Elle dispose des fonctions pour charger et sauvegarder des listes utilisateur au disque
- Elle contrôle les classes qui s'occupent du téléchargement d'images, et ignore celles qui existent déjà dans le cache.

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 :

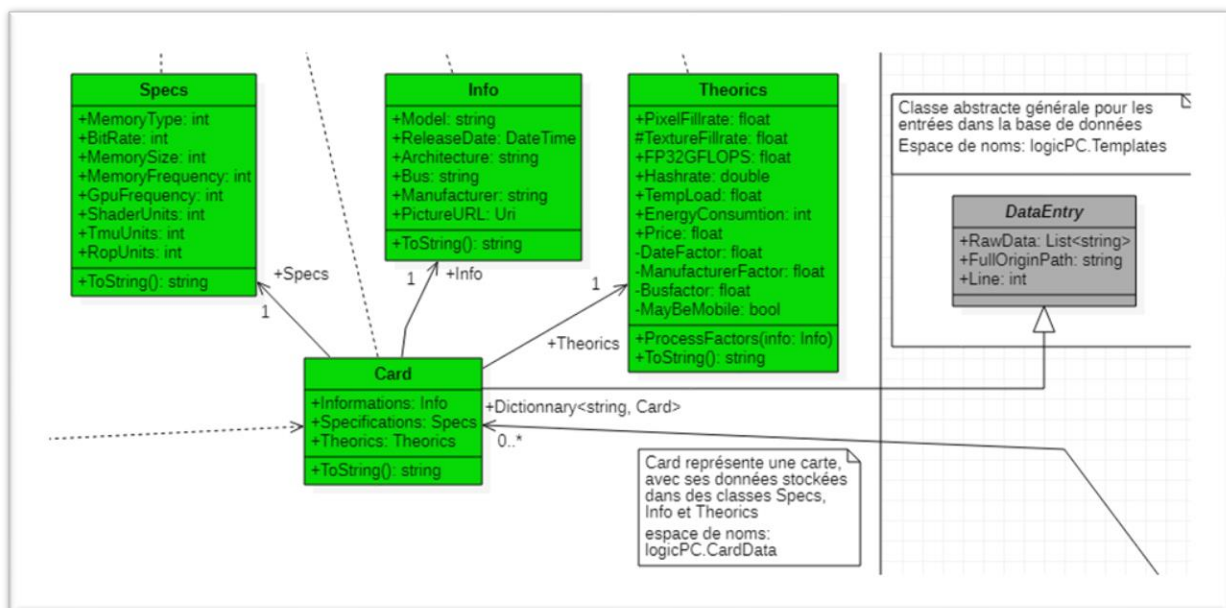


Dans ce cas, le sujet est `GestionnaireListes` (`MesListesUtilisateur` plus spécifiquement), l'observateur est **PropertyChanged**. L'évènement est lancé dès qu'une des méthodes de gestion de listes se termine.

Cette classe contient également une stratégie Persistance : `IPersistanceManager`

Voir l'annexe 1 pour plus de détails.

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.

²A terme, les données de cette classe seront largement utilisées pour pallier au 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 utilisées pour créer des graphiques permettant de voir les relations puissance/consommation/temps/coût/revenu.

RESPONSABILITES DE LA CLASSE `DATAENTRY`

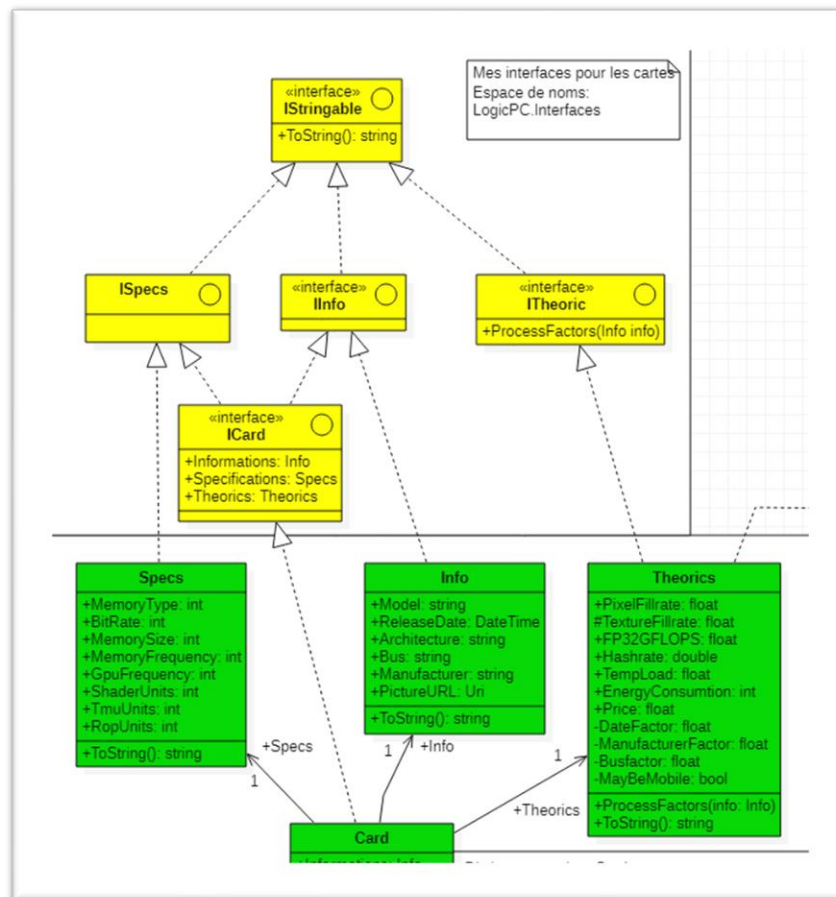
La classe abstraite `DataEntry` contient les données brutes récupérées par les classes d'importation de fichier. Le but réel de cette classe est double :

- 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 position 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.

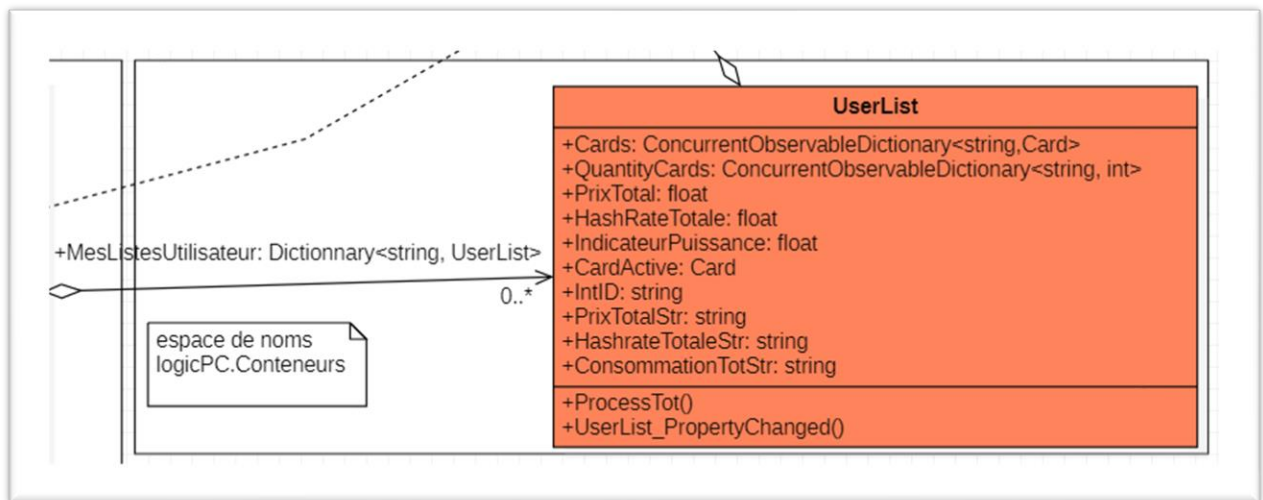
ITheoric :

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

ICard :

Une classe doit avoir les propriétés **Informations**, **Specifications**, et **Theorics**.

ESPACE DE NOMS LOGICPC.CONTENEURS

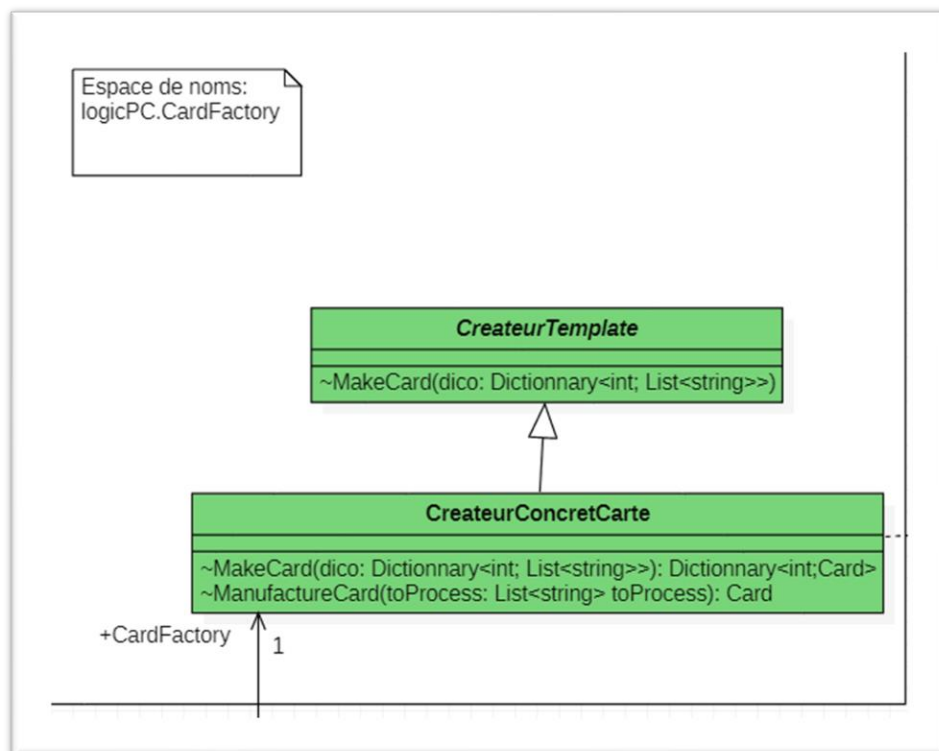


L'unique classe de cet espace de noms, [UserList](#), est une classe de stockage de données. Elle contient une liste de cartes graphique créé par l'utilisateur. Ses champs sont :

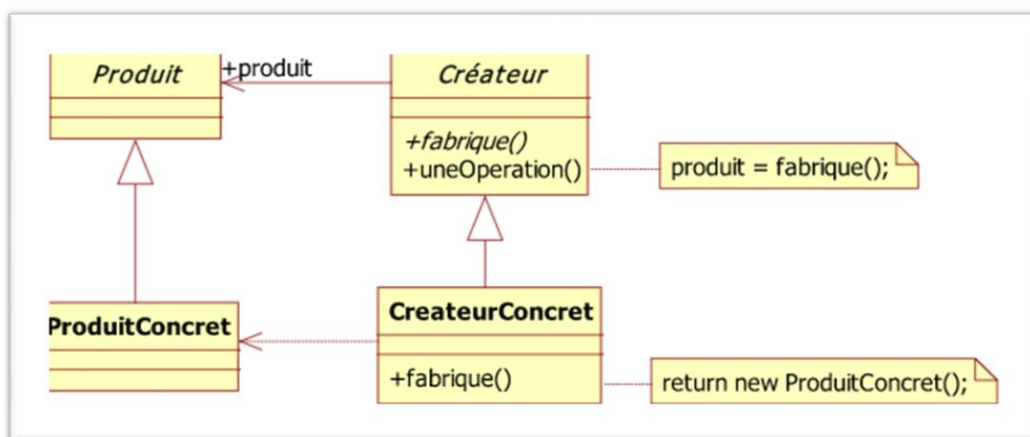
- [Cards](#) : Un dictionnaire de cartes qui contient toutes les cartes de la liste
- [QuantityCards](#) : Un dictionnaire constitué des mêmes clés que [Cards](#), avec en valeur le nombre de cartes pour cette clé.
- [PrixTotal](#) : Le coût direct total pour l'achat de toutes les cartes de cette liste.
- [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`.

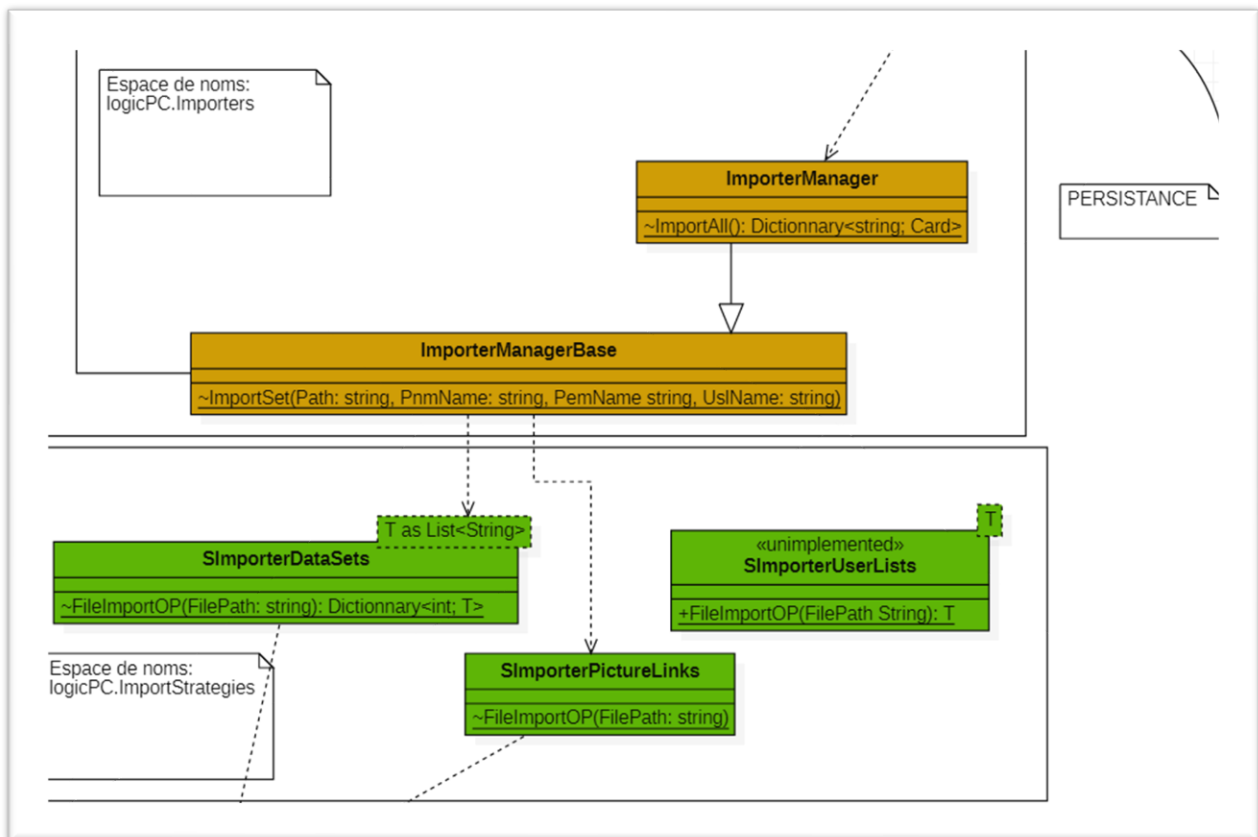
La particularité de cette Factory est qu'elle ne renvoie pas qu'une seule Card en prenant un seul string en argument. Elle prend un dictionnaire de listes de strings, et renvoie un dictionnaire de cartes.

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

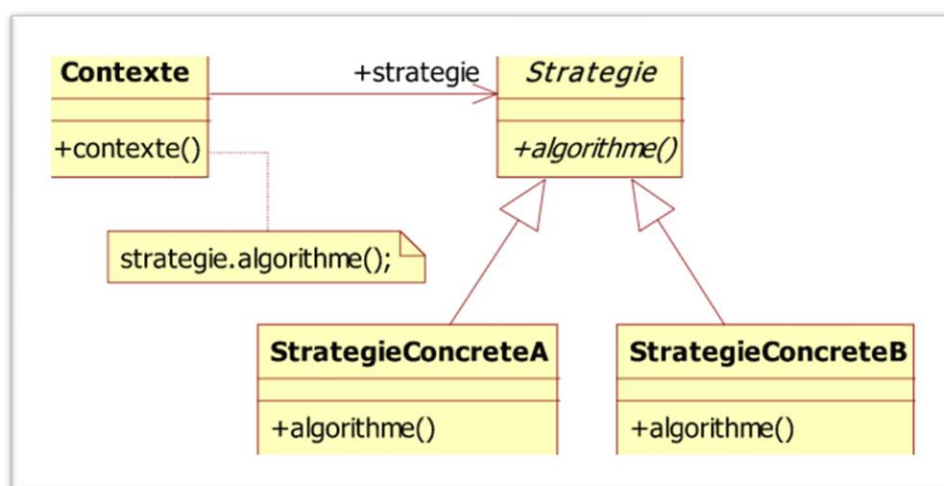
ANNEXES 1 – CLASSES DE PERSISTANCE

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

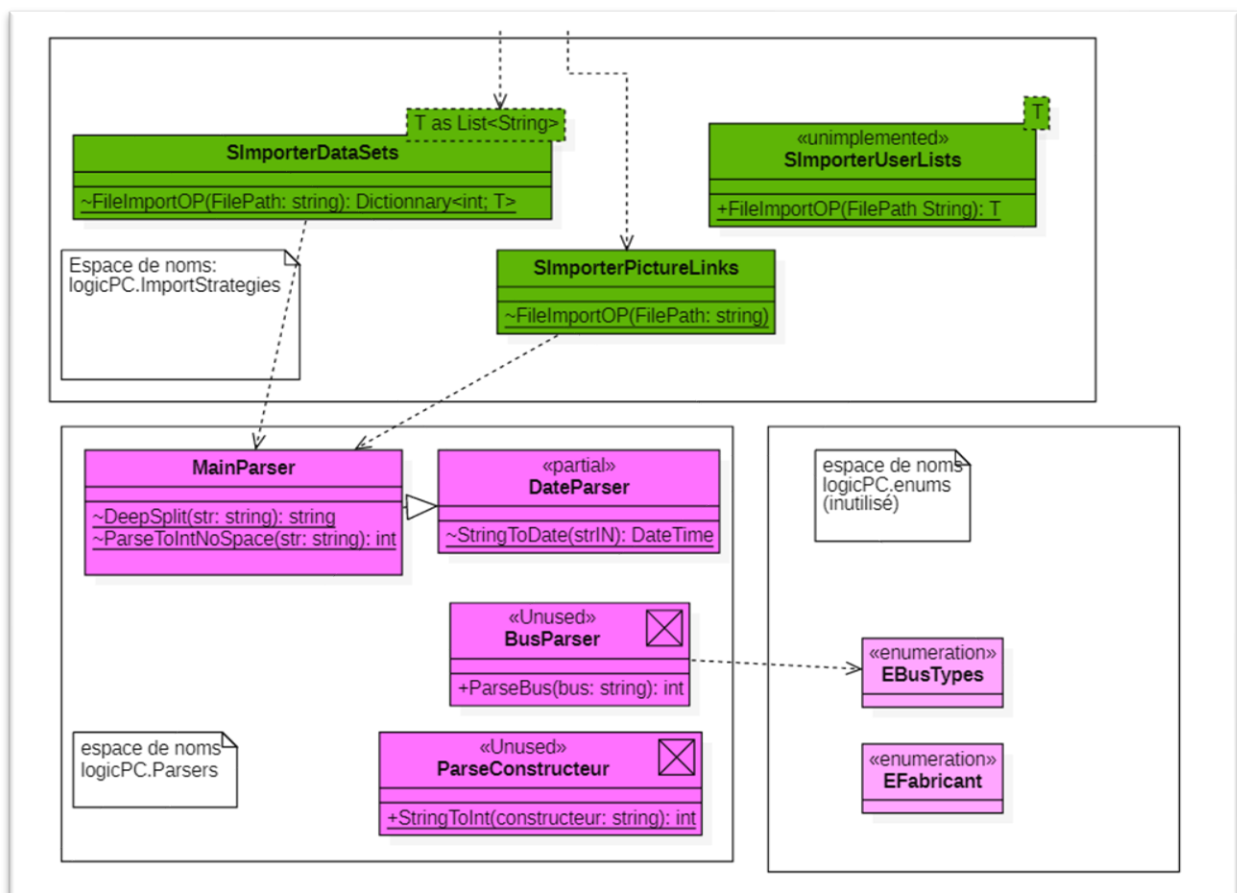


A l'origine les classes de cet espace de noms suivaient un patron de conception de stratégies (d'où leur nom) :



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.

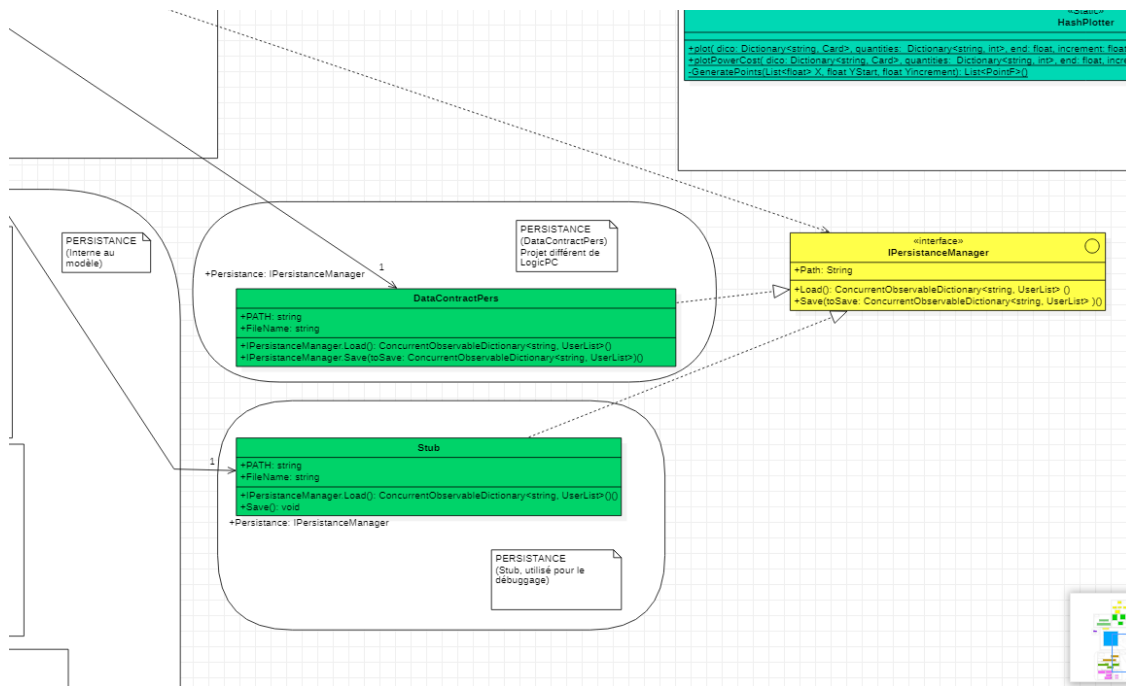


ImporterManagerBase instancie Une CardFactory. Une fois le processus d'importation et de traitement terminé, un dictionnaire<string, Card> est rendu à la classe apellante (ici GestionnaireListes).

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.

ESPACE DE NOMS PERSISTANCE

Dans un autre projet que logicPC et PCGUI se trouvent les classes de persistance XML et STUB. Ces classes servent à sauvegarder le contenu créé par l'utilisateur (donc différent du cache d'images dans PCGUI et de l'importation de données de logicPC qui sont des données internes au système).



On ne s'attardera pas sur le STUB, car il n'est utilisé que pour le debug. La classe `DataContractPers[istance]` adopte l'interface `IPersistenceManager`, et est instanciée dans le gestionnaire en tant que stratégie de persistance à appliquer par défaut. Elle utilise le `DataContract` pour stocker les informations sous forme XML sur le disque.

PERSISTANCE DU CACHE

Enfin, la dernière forme de persistance, le cache. Il est situé sur les vues (à cause de la classe `BitmapImage` qui n'est disponible qu'en WPF). Je ne vais pas détailler dessus car elle est assez simple :

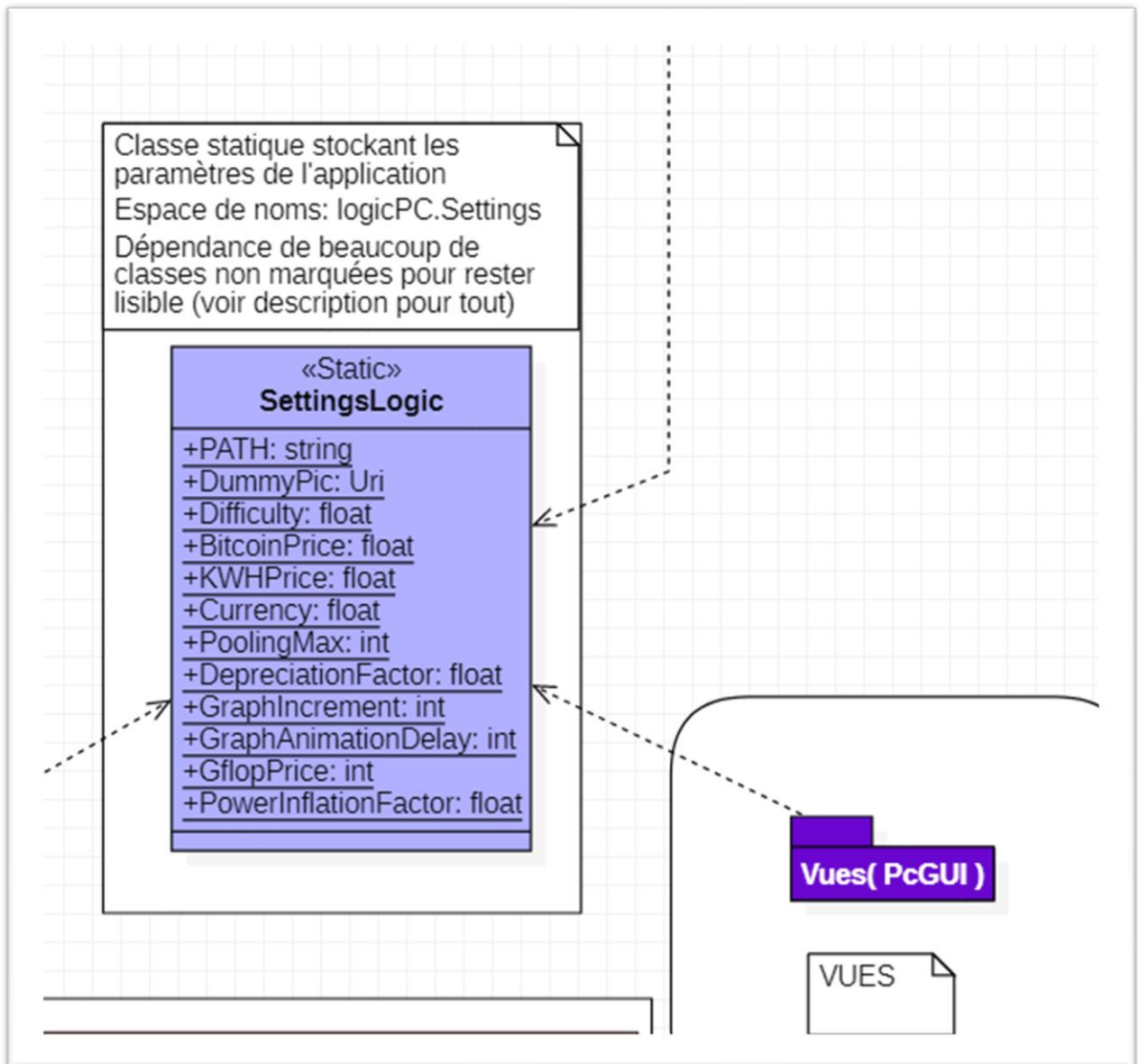
Une fois qu'une `bitmapImage` a fini de télécharger, elle fait avancer le tick de téléchargement de 1. Une fois que le tick dépasse un seuil donné (3 par défaut), les images sont affectées à leurs cartes respectives. Ce tick est mis en place pour éviter des problèmes de performance sur certaines machines.

Lors de l'affectation, le système vérifie si une image au format PNG existe dans le dossier cache avec le string d'identification unique du GPU qu'elle est supposée représenter (la clé du dictionnaire). Si ce n'est pas le cas, l'image est convertie en bitmap standard et on appelle `Bitmap.save()`, on la sauvegarde.

Le gestionnaire, lors du démarrage charge toutes les bitmaps présentes dans le dossier cache dans son dictionnaire de streams sous forme de streams de fichiers (normalement les images sont des streams http, mais cette manipulation fonctionne).

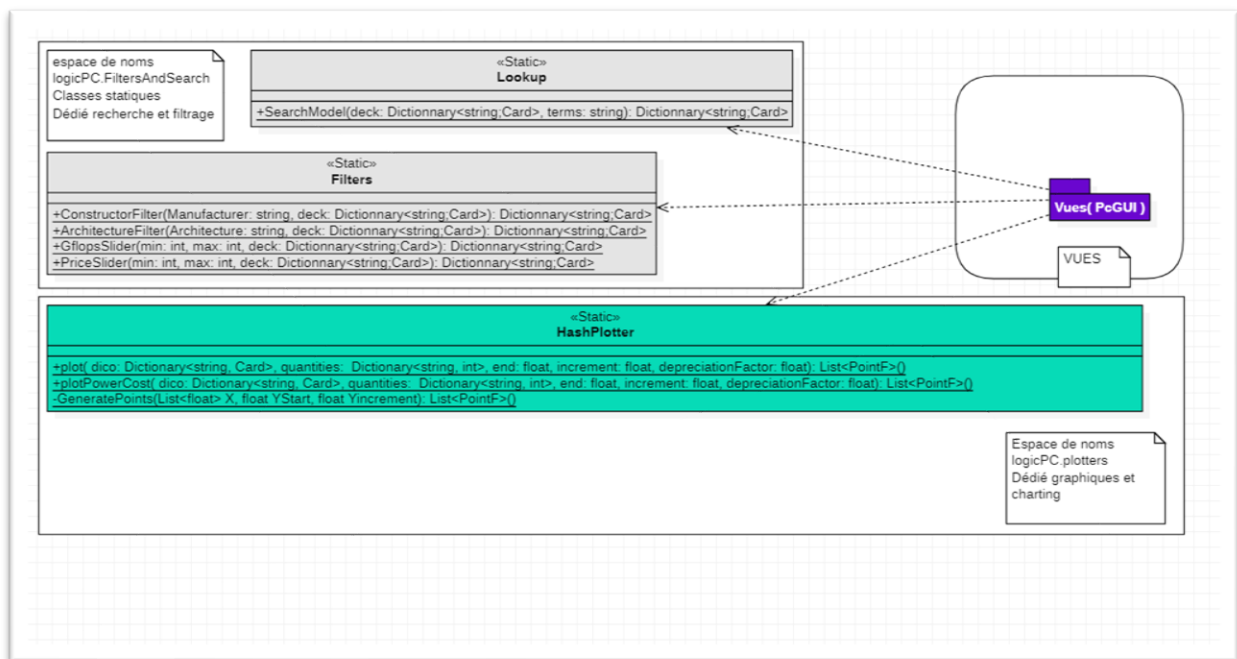
ANNEXE 2 – CLASSES STATIQUES INDEPENDANTES

LOGICPC.SETTINGS



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

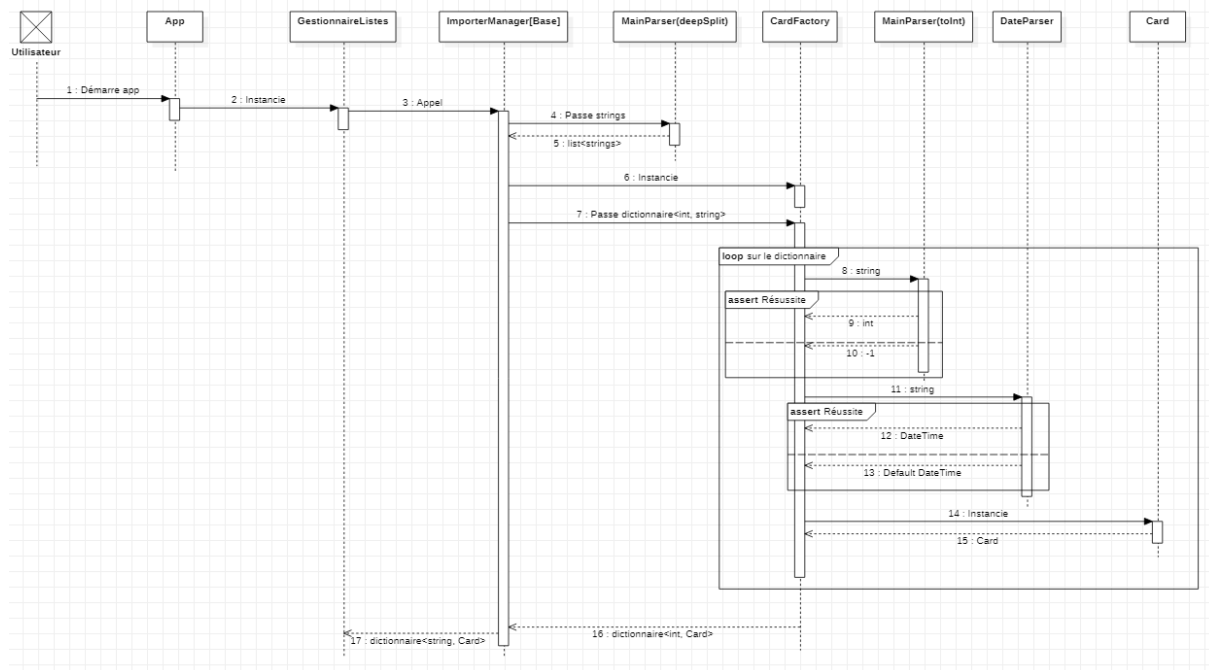
ANNEXE 3 – CLASSES STATIQUES « EXTERNES »



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 :

