PharmApp

Université de Bordeaux

Jérémy Morin
Louis Laine

Contents

A propos du rapport

Licence

A propos des auteurs

Jérémy Morin : **Etudiant en Licence 3 MIAGe** : *jer.morin@free.fr*

Louis Lainé : **Etudiant en Licence 3 MIAGe**

A l’attention

A l’attention de Madame Penois-Pinault

Github link

Cahier des charges

A la lecture du cahier des charges, nous avons déterminés quelles étaient les spécifications et les attentes.

L'application devra couvrir la gestion d'une base de données minimaliste de médicaments et effets correspondants.

Elle devra répondre aux spécifications suivantes :

Chargement des données

- La persistance des données sera assuré par un fichier **texte**.
- Les données devront être chargés, une fois parsées dans l'application durant toute la durée de vie de l'application.

Création d'une fonction d'usage

Une fonction d'usage devra être implémenté au cas ou l'utilisateur, n'a pas appelé le programme avec les bons paramètres.

Une fonctionnalité d'aide, accessible via un **-h** ou **-help**.

Validité de la base de données

Une exception devra être lancé, au cas ou la base de données n'est plus accessible.

Histogramme

Nécessité de créer, un histogramme des effets secondaires existant correspondant à un médicament et de pouvoir visualiser cet histogramme.

Recherche d'informations

L'application devra être en mesure, de faire plusieurs recherches sur la base de données.

- La liste des médicaments ayant un effet secondaire, en commun. L'utilisateur saisira un effet secondaire, l'application lui renverra l'ensemble des médicaments contenant cet effet.
- La liste des médicaments partageant le plus d'effet secondaires possibles avec un médicament saisi par l'utilisateur.
- L'utilisateur aura accès à tous les médicaments partageant un ou plusieurs effets secondaires. Pour ce faire l'utilisation de deux algorithme, qui devront être développé :
  - Parcoure tous les médicaments et sélectionne ceux qui ont le plus d'effet.
  - Renvoie les deux premiers résultats trouvés avec au moins un effet secondaire identique.

Bruit et précision

Calculer l'efficacité de l'algorithme de recherche par le biais de : `nbMedicamentsTrouves \ nbMedicamentsNormalementAtribues`↩

Ajout de nouveau médicament

L'utilisateur doit pouvoir ajouter un nouveau médicament ##Interface graphique L'application sera livrée avec une interface graphique en plus de l'interface classique.

Identification des ressources

Une fois le cahier des charges bien déchiffré, nous avons du déterminer le moyen par lequel les contraintes allaient être mise en oeuvre. Pour des raisons de flexibilité et d'évolution, nous avons décidés de développer en suivant une architecture assez spécifique. Cette architecture va permettre entre autres, de pouvoir changer aisement de source de fichier.

Actuellement il est nécessaire d'utiliser un fichier texte, mais si par exemple, on change pour du XML ou du JSON il suffira uniquement de recoder la classe Parser en suivant l'interface **IParser**. Ainsi la source de donnera changera, sans affecter le reste de l'application et restera totalement invisible pour l'utilisateur.

L'utilisation des containers classique C++ a été utilisé pour développer cette application.

Fonctionnement

Conception

Lors de la phase de conception nous avons décidés de mettre en place une architecture trois-tiers.

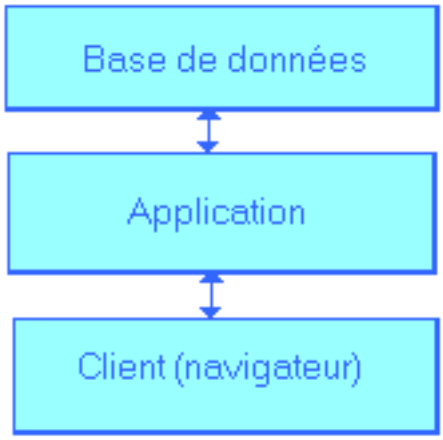


Figure 1: Schema de l'architecture trois-tiers

- **Présentation correspondant** à l'affichage, le dialogue avec l'utilisateur.
- **Traitement correspondant** à la couche métier, la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative
- **Accès aux données** correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

En suivant cette logique, nous avons donc conçu l'architecture suivante concernant l'application.

- La couche de présentation est représenté par les modules **mainTerm** et **mainGui**
- La couche de traitement est représenté par les packages **Abstract Parser** et **Dll**, représentant la logique applicative, ainsi que l'accès à la base de données.
- Les données, correspondent au fichier **liste.txt**

Diagramme UML en annexe

Logique métier

ParserFichier

```
34. typedef std::map<string, std::vector<string> > mapParse;
35. typedef std::map<string, std::vector<string> >::iterator mapParseIterator;
```

```
36. typedef std::vector<string> leFichier;
```

C'est la classe qui va permettre la dialogue entre le fichier texte et les objets.

Cette classe va charger les données issue du fichier texte, dans une *mapParse*.

Cette map sera constitué en clé, d'un *string* associé avec un *vector*. Cette conception rappelle la base de données qui à la forme suivante :

```
nom du médicament : effet1, effet2 et effet3.
```

ObjectFactory

```
34. typedef std::map<int, Medicament> dataMap;
35. typedef std::map<string, std::vector<string> > parserMap;
```

ObjectFactory, est la classe permettant la transition des données en objet.

Cette couche, abstraite permettre de remonter les données en objet vers la couche supérieure.

La fonction *transformToObject()*, qui se base sur l'utilisation du *ParserFichier*, va parcourir chaque ligne du fichier et créer les objets correspondant, à l'aide des fonctions de *ParserFichier* et de la Dll pour créer les objets *Médicament* et *Effet*

DataProvider

```
20. typedef std::map<int, Medicament> dataMap;
...
24. typedef std::map<string, std::vector<string> > parserMap;
...
28. typedef std::map<string, double> dataEffet;
```

La couche DataProvider, héritant directement de la couche inférieure *ObjectFactory*, sera considéré comme l'objet **base de données** de l'appliaction.

Les données sont durant toute la durée de vie de l'appliaction chargé dans cet objet via un attribut privée *DataProvider::dataStorage*.

Ainsi toutes les fonctions de traitements concernant les données utiliseront le *DataProvider* comme source de données.

Utilisation

Lors de l'instanciation de l'objet (équivalent à la création de la base de données) la méthode mère *TransformToObject()* sera appelé pour pour récupérer dans le scope les données et hydrater l'attribut *DataProvider::dataStorage*. \

```
try{
    this->dataStorage = this->transformToObjet();
}catch(ObjectFactory::FactoryException e){
    e.what();
}
```

Toutes les “requêtes”, utiliseront cet attribut comme source de données.

Controller

C’est la couche d’abstraction pour la gestion des évenements. \ Pour éviter de créer une fonction *main()* à ralonge et illisible, cette classe se chargera de renvoyer en fonction d’un potentiel choix utilisateur, les données mise en forme qui pourront être affichés dans la fonction *main()*, par la suite.

Cette classe se voulant polyvalente pour une utilisation graphique ou cli, renverra en fonction d’attribut passé en paramètre dans les méthodes de, renvoyer un affichage de données ou de renvoyer les données via un **dataMap**.

```
//Exemple d'une fonction
template <typename T>
T Controller::dispEffetFromMedoc(...) {
    //Fonction d'usage de la base de données
    try{
        this->tabTemp = this->database.getMapMedocFromEffet(theNomEffet);
    }catch(DataProvider::DBException e){
        e.getMessage();
    }
    ...
    //Le renvoie utile pour l'utilisation dans un main graphique par exemple.
    if (asReturn){
        return this->tabTemp;
    }
    ...
    return T();
}
```

Ainsi, qu’on développe un main graphique ou cli, la classe controller sera toujours celle qui sera utilisé pour la gestion des évenements, grâce à l’utilisation de *template* (concept C++).

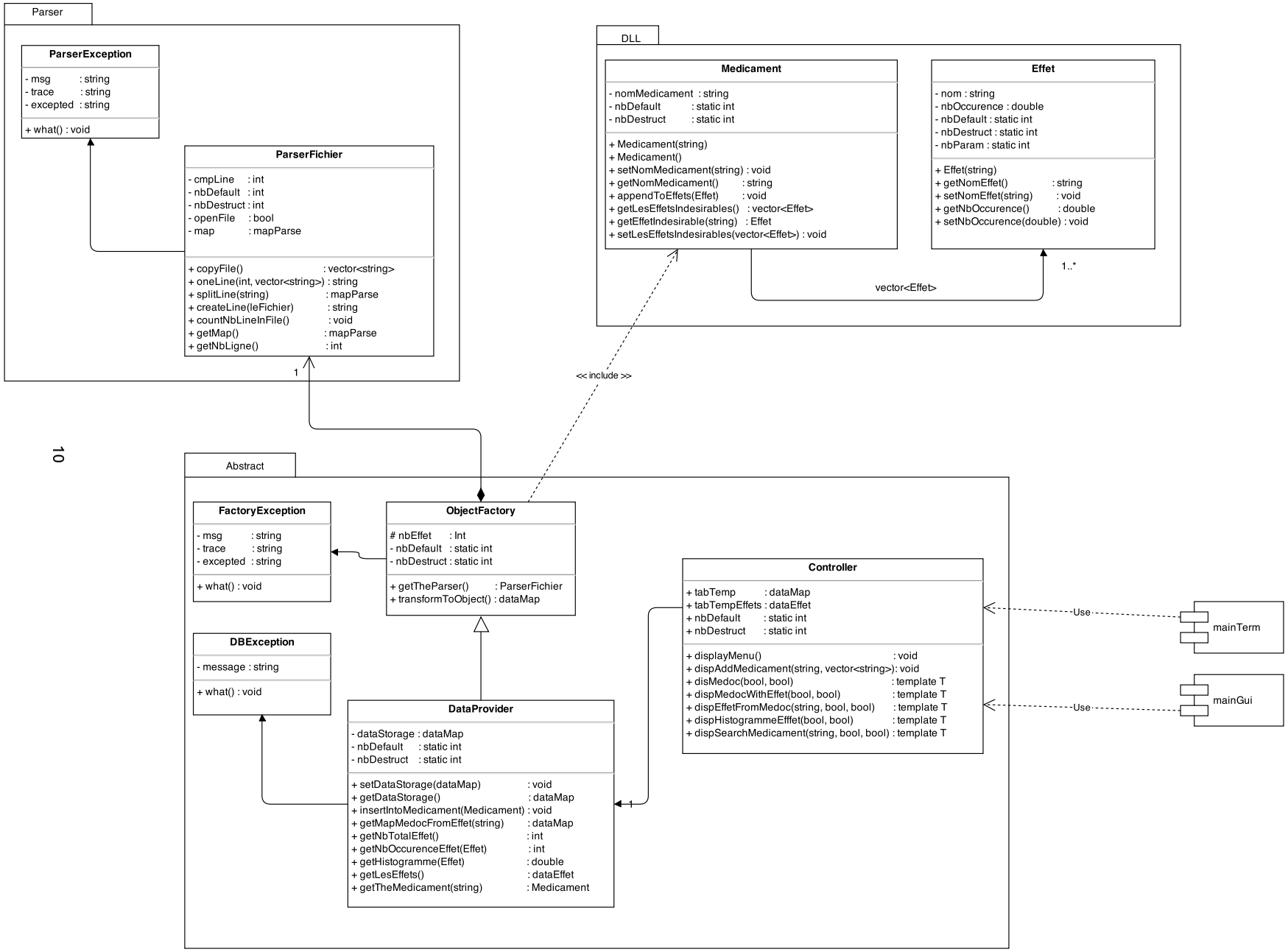
Interface Graphique

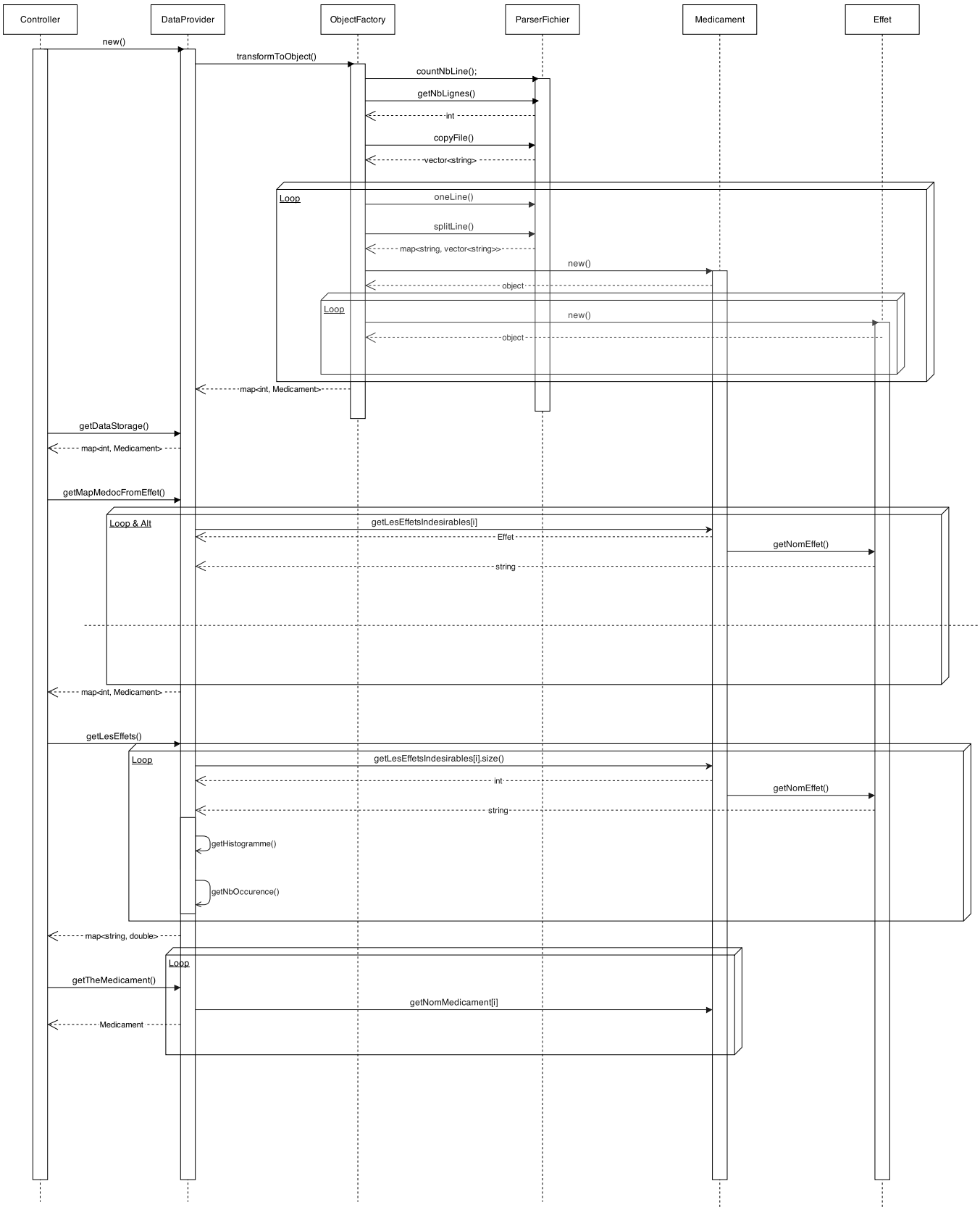
Le cahier des charges préconisait une utilisation de QtCreator.

Par manque de souplesse et les trop fortes contraintes imposés par Qt, nous avons décidé de ne pas utiliser cette libraire et d'utiliser une beaucoup plus puissante, **GTK** à l'aide de **GTKmm**.

L'application n'a malheureusement pas été développé jusqu'à terme (à retrouver sur la **branch UI** du repo).

Cependant, elle prend en compte la fonctionnalité de visualisation des médicaments avec les effets, ainsi que la recherche de médicaments ayant un effet en commun.


Parser
ParserException
- msg : string
- trace : string
- excepted : string
+ what() : void
ParserFichier
- cmpLine : int
- nbDefault : int
- nbDestruct : int
- openFile : bool
- map : mapParse
+ copyFile() : vector<string>
+ oneLine(int, vector<string>) : string
+ splitLine(string) : mapParse
+ createLine(leFichier) : string
+ countNbLineInFile() : void
+ getMap() : mapParse
+ getNbLigne() : int
1
DLL
Medicament
- nomMedicament : string
- nbDefault : static int
- nbDestruct : static int
+ Medicament(string)
+ Medicament()
+ setNomMedicament(string) : void
+ getNomMedicament() : string
+ appendToEffets(Effet) : void
+ getLesEffetsIndesirables() : vector<Effet>
+ getEffetIndesirable(string) : Effet
+ setLesEffetsIndesirables(vector<Effet>) : void
Effet
- nom : string
- nbOccurence : double
- nbDefault : static int
- nbDestruct : static int
- nbParam : static int
+ Effet(string)
+ getNomEffet() : string
+ setNomEffet(string) : void
+ getNbOccurence() : double
+ setNbOccurence(double) : void
1..*
vector<Effet>
<< include >>
Abstract
FactoryException
- msg : string
- trace : string
- excepted : string
+ what() : void
ObjectFactory
nbEffet : Int
- nbDefault : static int
- nbDestruct : static int
+ getTheParser() : ParserFichier
+ transformToObject() : dataMap
DBException
- message : string
+ what() : void
DataProvider
- dataStorage : dataMap
- nbDefault : static int
- nbDestruct : static int
+ setDataStorage(dataMap) : void
+ getDataStorage() : dataMap
+ insertIntoMedicament(Medicament) : void
+ getMapMedocFromEffet(string) : dataMap
+ getNbTotalEffet() : int
+ getNbOccurenceEffet(Effet) : int
+ getHistogramme(Effet) : double
+ getLesEffets() : dataEffet
+ getTheMedicament(string) : Medicament
1
Controller
+ tabTemp : dataMap
+ tabTempEffets : dataEffet
+ nbDefault : static int
+ nbDestruct : static int
+ displayMenu() : void
+ dispAddMedicament(string, vector<string>) : void
+ disMedoc(bool, bool) : template T
+ dispMedocWithEffet(bool, bool) : template T
+ dispEffetFromMedoc(string, bool, bool) : template T
+ dispHistogrammeEfffet(bool, bool) : template T
+ dispSearchMedicament(string, bool, bool) : template T
Use
mainTerm
Use
mainGui


Controller
DataProvider
ObjectFactory
ParserFichier
Medicament
Effet
new()
transformToObject()
countNbLine();
getNbLignes()
int
copyFile()
vector<string>
Loop
oneLine()
splitLine()
map<string, vector<string>>
new()
object
Loop
new()
object
map<int, Medicament>
getDataStorage()
map<int, Medicament>
getMapMedocFromEffet()
Loop & Alt
getLesEffetsIndesirables[i]
Effet
getNomEffet()
string
map<int, Medicament>
getLesEffets()
Loop
getLesEffetsIndesirables[i].size()
int
getNomEffet()
string
getHistogramme()
getNbOccurence()
map<string, double>
Loop
getTheMedicament()
getNomMedicament[i]
Medicament