Rapport de projet

Programmation orientée objet

CHENET Dorian – CHOLLET Alexis

Table des matières

1 Planning et répartition des tâches

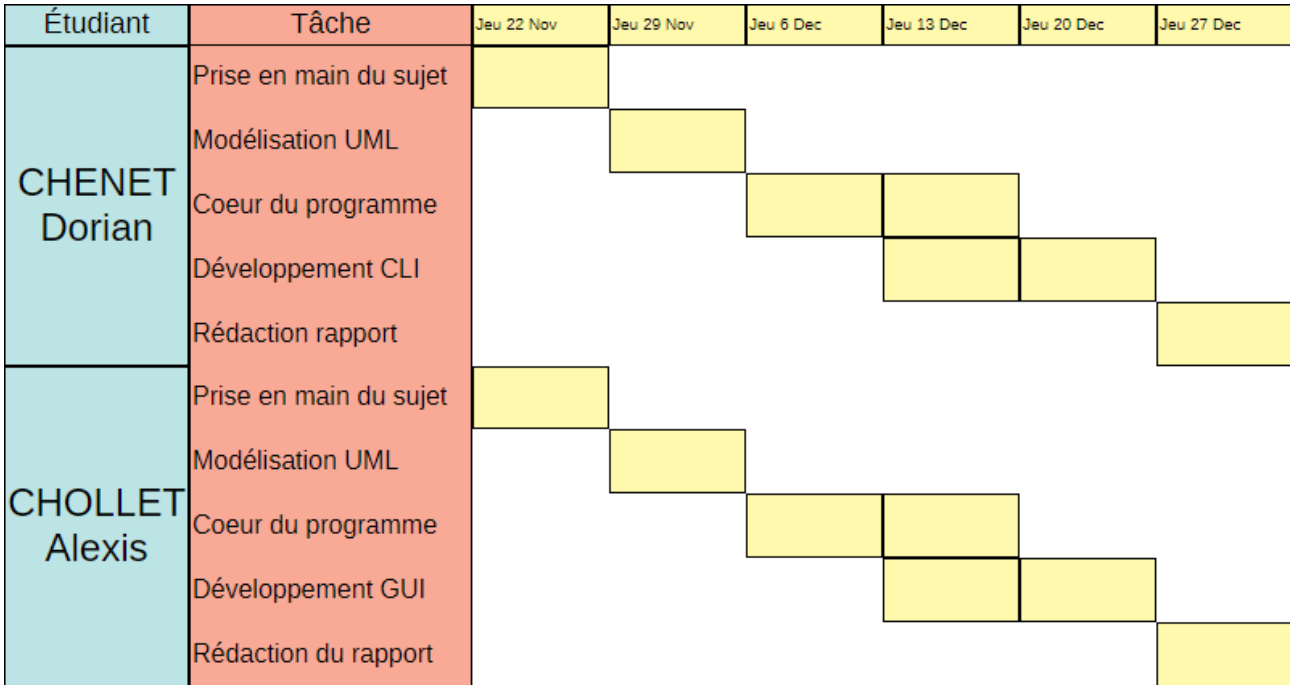


Illustration1: Gantt du projet

Planning

Comme vous pouvez le voir sur l' Illustration1: Gantt du projet, nous avions opté pour une répartition des tâches de manière commune dans un premier temps, puis de manière parallèle dans un second temps. La répartition des tâches dans le temps a bien été respectée sauf pour la modélisation UML qui s'est faite pendant la semaine du 24 au 27 pendant rédaction du rapport car des changements dans la structure du programme ont été réalisés au cours du développement du GUI. De plus, deux éléments non présents dans le gantt: le fichier readme.txt et la javadoc on été écrits pendant la semaine du 17 au 23. Enfin, à titre indicatif, nous avons travaillé conjointement 3 jours en dehors des heures de cours: Lundi 17, Jeudi 20 et Vendredi 21 Décembre.

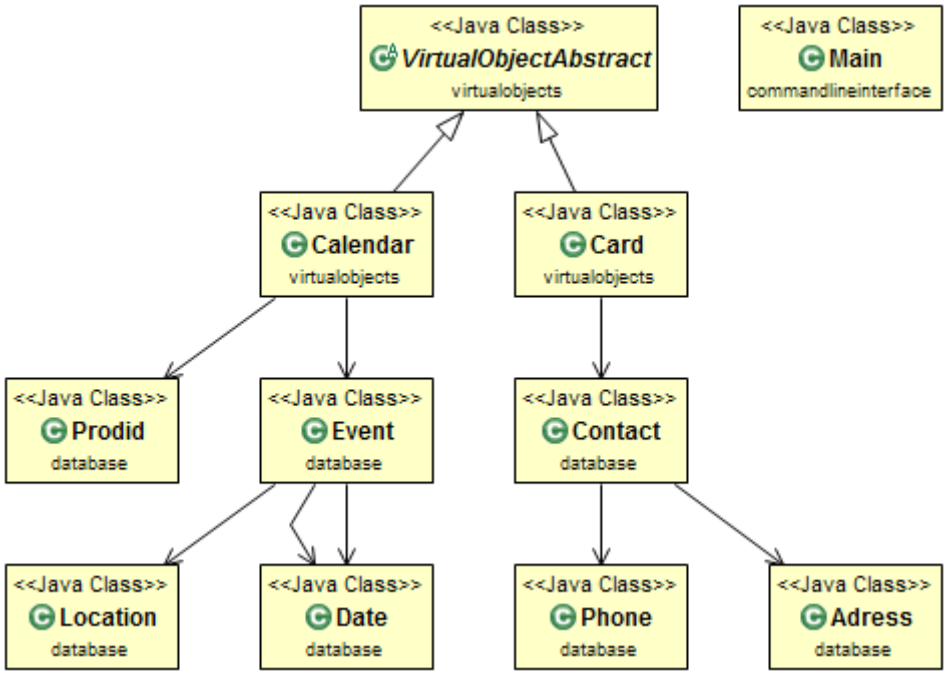
Répartition des tâches

Tâche à réaliser	Etudiant(s)
CLI	CHENET
GUI	CHENET, CHOLLET
Javadoc	CHENET, CHOLLET
Readme.txt	CHOLLET
Rapport	CHENET

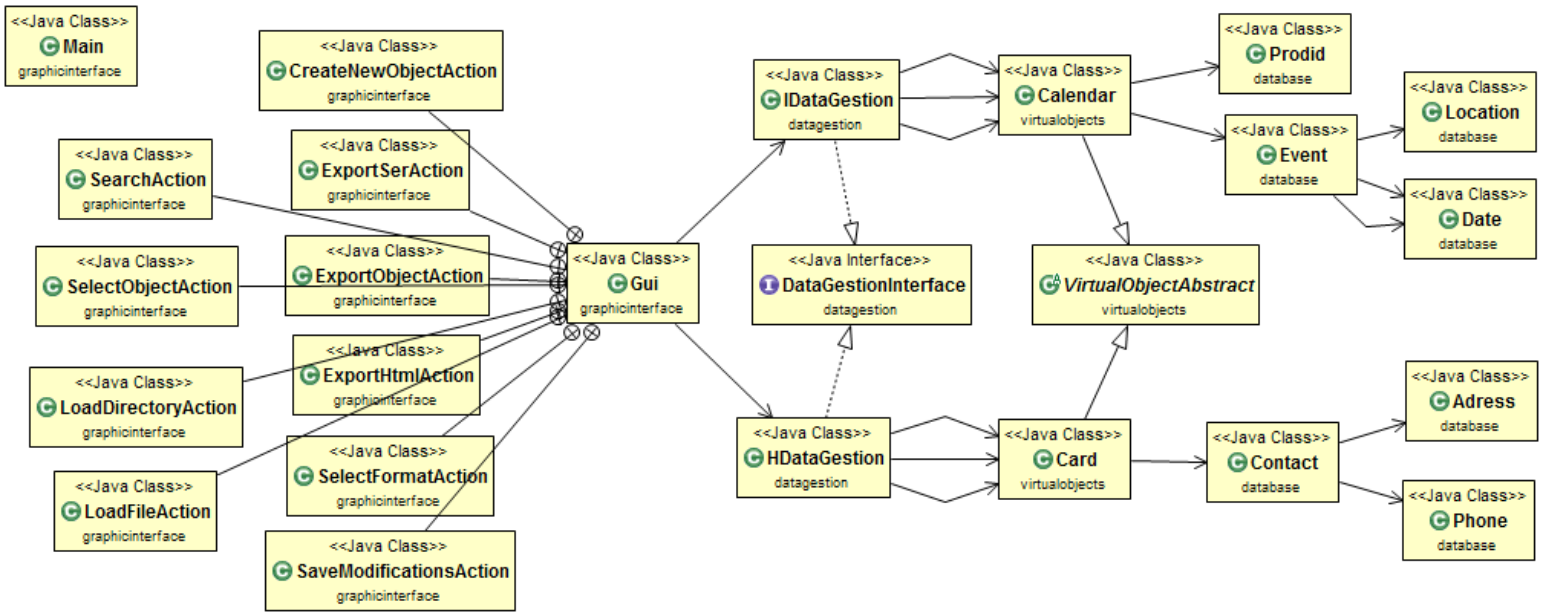
Dans chaque classe du CLI et GUI est notifié qui a écrit la javadoc et qui a écrit la classe.

2 Diagrammes de classes UML[1]

CLI



Gui



1 Les diagrammes complets spécifiant attributs et méthodes sont fournits dans le dossier.

3 Structure de stockage de données

Classes "de données" (package database et virtualobjects)

Branche vCard

Classe	Fonction	Composée d'objet(s) de classe
Card	Générer des fichiers vCard et HTML à partir d'un contact.	Contact
Contact	Apporte à la classe Card les informations sur la personne qui sera l'objet de la vCard.	Phone, Adress,String
Phone	Contient un numéro de téléphone de la personne et le lieu auquel le numéro est associé (attibut place).	String
Adress	Contient les informations relatives à un lieu et ce qu'il représente (attibut type), par exemple: maison, travail...	String

Branche hCalendar

Classe	Fonction	Composée d'objet(s) de classe
Calendar	Générer des fichiers hCalendar et HTML à partir d'un contact.	Event, Prodid
Event	Apporte les informations sur l'événement lui-même. Cette classe contient deux dates pour la date de début et la date de fin de l'événement.	Date (2), Location
Date	Contient une date.	String
Location	Contient les informations sur le lieu de l'événement.	String
Prodid	Apporte les information concernant l'organisateur de l'événement.	String

Classe abstraite VirtualObject

Mutualise pour les fichiers les classes Card et Calendar les procédures d'export de fichiers texte (utilisée pour les formats ics/vcf/html) et de fichiers binaires (.ser) car elles fonctionnent de la même manière pour ces deux classes. Les classes Card et Calendar héritent donc de VirtualObjects.

Implémentation pour la partie CLI

Commençons par rappeler les contraintes du sujet, la CLI doit avoir comme fonctions:

- Lister tous les fichiers .vcf/.ics dans un répertoire donné.
- Prendre en entrée un fichier .vcf/.ics et l'afficher dans la console.
- Prendre en entrée un fichier .vcf/.ics et sauvegarder son contenu dans un fichier .ser.
- Prendre en entrée un fichier .vcf/.ics et générer sont fragment HTML correspondant.

Pour remplir ces fonction, la structure de donnée décrite au dessus avec une branche vCard (classe Card) et une branche hCalendar (classe Calendar) qui découlent de la classe VirtualObject est suffisante. Nous n'utilisons donc qu'une classe "Main" pour gérer les lignes de commande et les entrées/sorties de données.

Implémentation pour la partie GUI

Rappelons encore une fois les différentes tâches que le GUI doit effectuer:

- Exploration d'un arborescence quelconque permettant de lister uniquement les fichiers de type .vcf/.ics.
- La sélection d'un fichier provoque son affichage.
- Avoir la possibilité de modifier les principaux champs d'un fichier sélectionné et leur enregistrement.
- Pouvoir générer un fragment HTML correspondant au fichier sélectionné.

Là encore, la structure de données de base aurait suffit mais nous avons voulu aller plus loins en implémentant une base de donnée dans laquelle on pourait charger un certain nombre de fichiers, pouvoir naviguer dans cette dernière et faire des recherches. Ceux afin d'essayer de donner un aspect plus pratique à nôtre logiciel. Voila donc la structure de gestion de données.

4 Structure de gestion de données

Classes "de gestion" de données (package datagestion)

Il y a deux classes de gestion dans la structure de l'application GUI: IDataGestion pour ICalendar et VDataGestion pour VCard.

Ces deux classes contiennent:

- 3 attributs: database pour stocker les informations des fichiers chargés, searchresults pour stocker les résultats d'une recherche et selected(card/calendar) pour garder en mémoire l'objet sélectionné afin de pouvoir travailler avec.
- Des méthodes qui permettent la gestion des données (recherche, lecture, écriture).

De plus, VDataGestion et IDataGestion implémentent toute les deux une interface DataGestionInterface. Cette interface n'est pas vraiment utilisée dans le programme, elle est surtout là pour montrer que les deux classes de gestion ont des similitudes, c'est un point à améliorer dont nous reparlerons.

5 Fonctionnement du CLI

Traitement des inputs par Swich Case

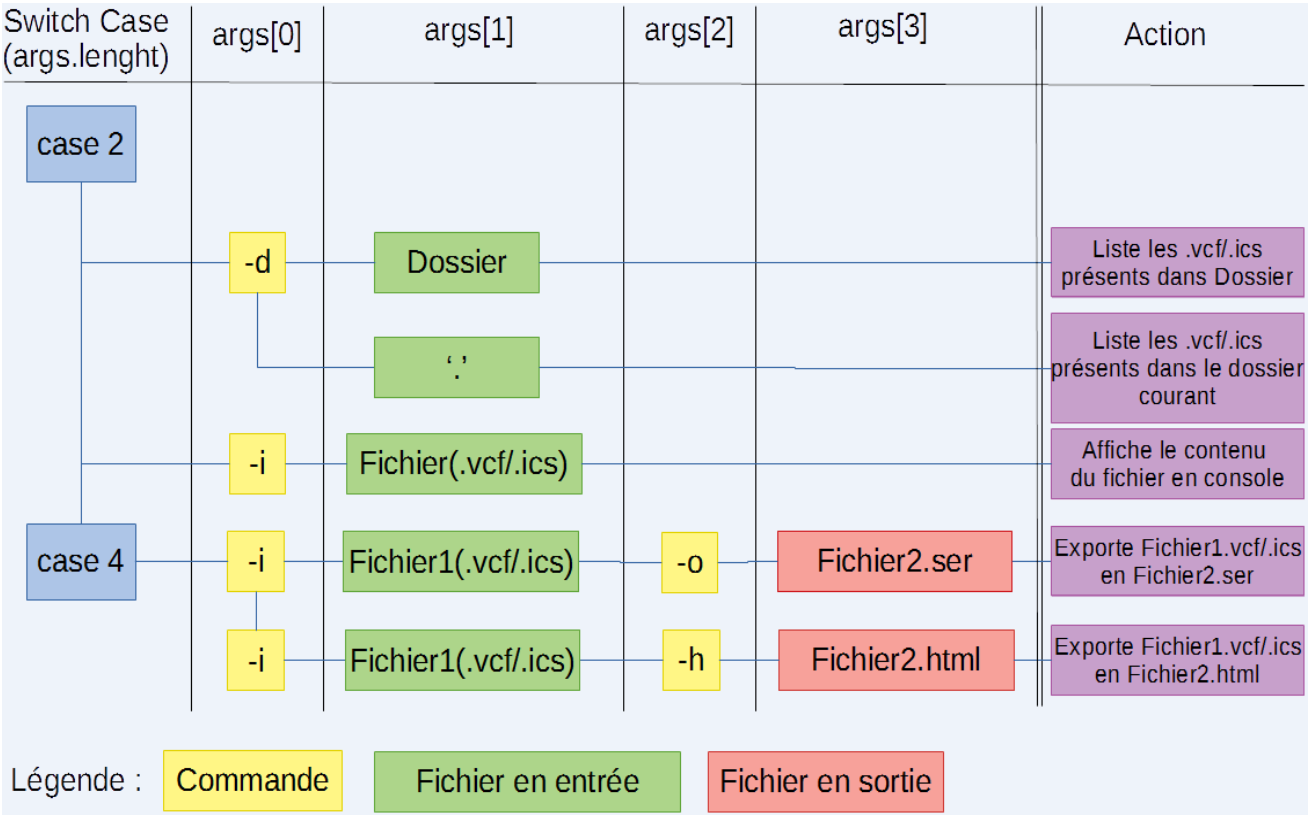
Un court rappel sur les commandes que la console doit gérer:

- -d "dossier": Liste dans la console tous les fichiers (.vcf/.ics) dans l'arborescence quelconque "dossier", si "dossier" est égal à ".", effectue la recherche dans le répertoire courant.
- -i "fichier(.vcf/.ics)": Afficher le contenu du fichier dans la console.
- -i "fichier1(.vcf/.ics)" -o "fichier2.ser": Sauvegarde le contenu de "fichier1" dans un fichier binaire "fichier2.ser".
- -i "fichier1(.vcf/.ics)" -h "fichier2.html": Exporte le fragment HTML correspondant à "fichier1" dans un "fichier2.html".

Comme nous l'avons vu, la CLI utilise simplement une structure "de stockage" de données ainsi que les classes Card, Calendar et VirtualObject qui permettent de gérer les entrées/sorties de fichiers.

La gestion de lignes de commandes et les appels de fonction se font donc dans le Main; les arguments entrés dans la console sont transmis au programme par le biais du tableau String[] args et sont traités par un Switch Case effectué sur la longueur du tableau args.

Voici un schema descriptif des actions réalisés par le Switch Case (la gestion d'erreurs n'y est pas représenté pour lisibilité).



Des messages d'erreur apparaissent lorsque la commande n'arrive au bout d'aucun de ces chemins. Si aucune commande n'est entrée, les commandes disponibles sont affichées en console grâce à la condition default du switch case.

Recherche de fichiers .vcf/.ics dans un dossier

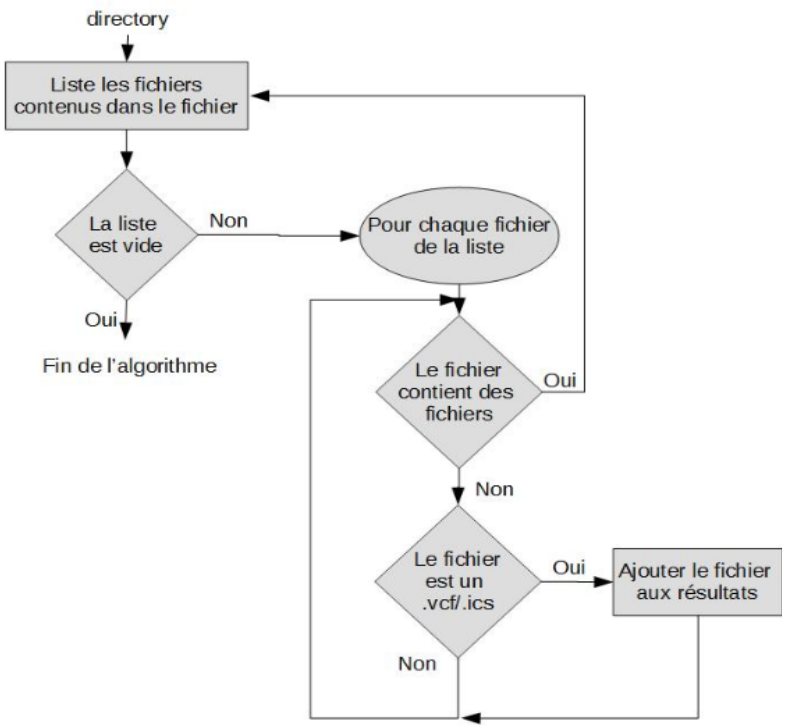


Illustration 2: Algorigramme de la procédure searchFile.

Dans le Swich Case, la recherche des fichiers .vcf/.ics se fait grâce à une inner class SearchDirectory qui possède comme attributs deux listes ArrayList<File>: searchresultsics et searchresultsvcf pour sauvegarder les fichiers trouvés. La recherche des fichier s'effectue via un méthode récursive: searchFile(File directory) dont vous pouvez voir un algorigramme en Illustration 2.

En dehors de l'inner class, on crée alors un objet SearchDirectory sf, auquel on donne en entrée le fichier que l'utilistateur a entré en commande.

Les résultats de cette recherche sont ensuite affichés dans la console.

Enfin, quand l'utilisateur donne en entrée le caractère '.', nous récupèrons l'emplacement courant du fichier .jar avec un objet ClassLoader (java.lang.ClassLoader) et utilisons la procédure searchFile.

Sérialiser un fichier .vcf/.ser et exporter un fragment HTML

Pour sérialiser un fichier .vcf/.ics, le schema est le suivant:

- Créer un nouvel objet Card / Calendar.
- Charger le fichier avec la procédure card.loadVcfFile / calendar.loadIcsCard.
- Sérialiser le fichier chargé avec la procédure card/calendar.exportSer.

De même, pour exporter un fichier .vcf/.ics, le schema est:

- Créer un nouvel objet Card / Calendar.
- Charger le fichier avec la procédure card.loadVcfFile / calendar.loadIcsCard.
- Sérialiser le fragment correspondant avec la procédure card/calendar.exportTextFile.

La procédure est nomée exportTextFile car nous l'utilisons dans la partie GUI pour exporter des fichiers .vcf/.ics car ils sont des fichiers texte au même titre que les fragments HTML.

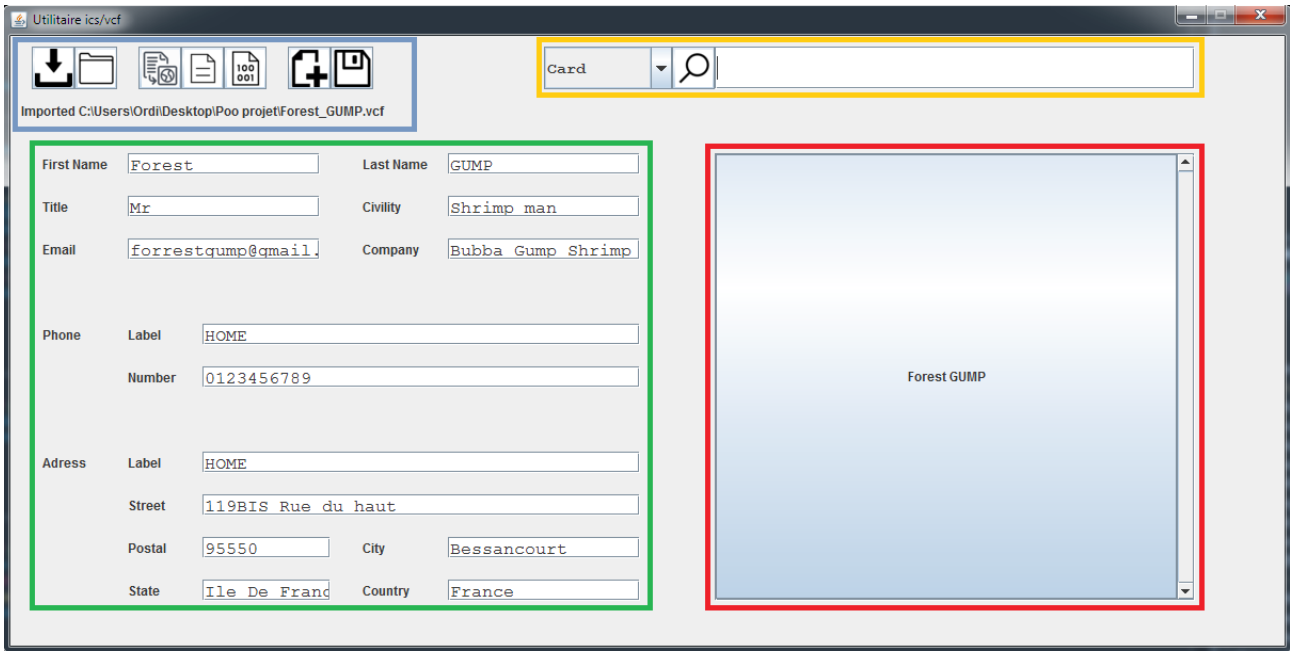
Nous passons donc à cette procédure le fragment généré par la fonction card/calendar.getHtmlFragment, pour pouvoir l'écrire dans un fichier texte.

Le fragment Html est généré en concaténant les toString() des différentes données (Adress,Phone/Prodid,Event...) dans une chaine de caractère ensuite retourné par la fonction.

6 Fonctionnement du GUI

Description de l'interface

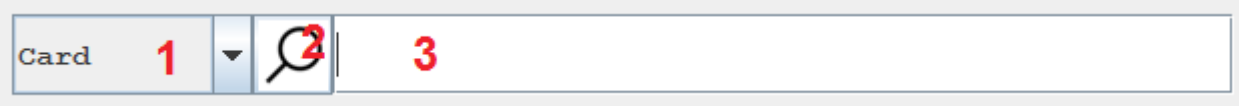
Dans l'interface graphique, on peu identifier 4 zones principales:



Panneau de commandes

1. Charger un fichier .vcf/.ics/.ser[2]
2. Charger un ensemble de fichiers .vcf/.ics (similaire à la commande -d du CLI).
3. Exporter le fragment HTML correspondant au fichier sélectionné.
4. Exporter le fichier sélectionné en .vcf/.ics.
5. Exporter le fichier sélectionné en .ser.
6. Editer un nouveau fichier.
7. Sauvegarder les modifications.

Panneau de recherche



1. Sélection du format à afficher / chercher.
2. Bouton recherche.
3. Barre de recherche.

Appuyer sur Enter dans la barre de recherche est équivalent à cliquer sur le bouton recherche (ce

2 Avant de charger un fichier vCard/iCalendar en .ser, il faut sélectionner le format Card/Calendar dans le menu déroulant dans le panneau de recherche.

dernier est là plus à titre indicatif pour montrer que la barre adjacente est un barre de recherche plus que pour réellement servir). La recherche s'effectue dans les bases de données VDataGestion ou IDataGestion en fonction du format sélectionné dans le menu déroulant 1.

Affichage des données d'un fichier

Les informations contenues dans un fichier s'affichent dans des zones de texte où elles penvent ensuite être modifiées, sauvegardés puis exportées dans un fichier (.vcf/.ics) ou encore .ser ou .html.

Affichage de la base de données

Dans cette zone apparaît la liste de tous les documents chargés dans la base de données, il est possible de dérouller la liste quand elle est trop longue pour être affichée entièrement gâce à la barre de scorll sur le côté. Chaque élément qui apparaît dans cette liste est cliquable ce qui a pour effet d'aficher son contenu dans la partie gauche de la fenêtre.

Composants graphiques

Schematique des composants

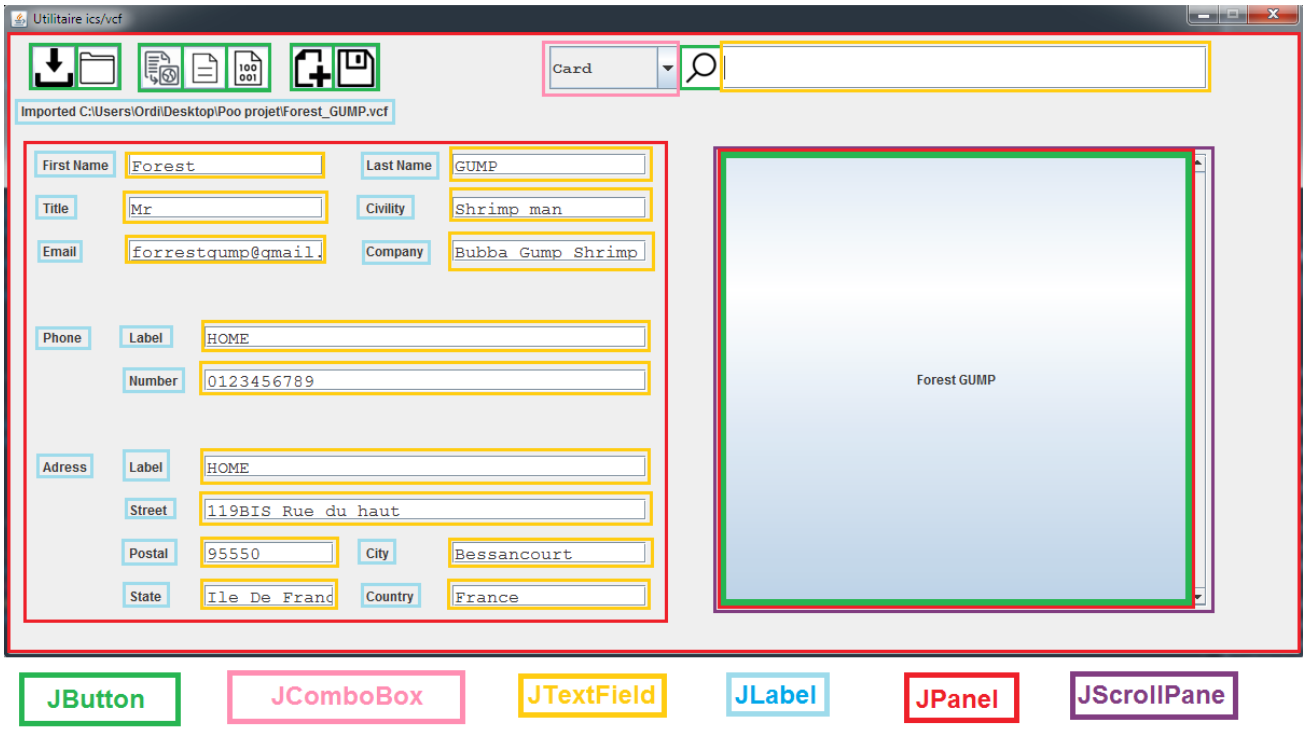


Illustration 3: Description des composants graphiques

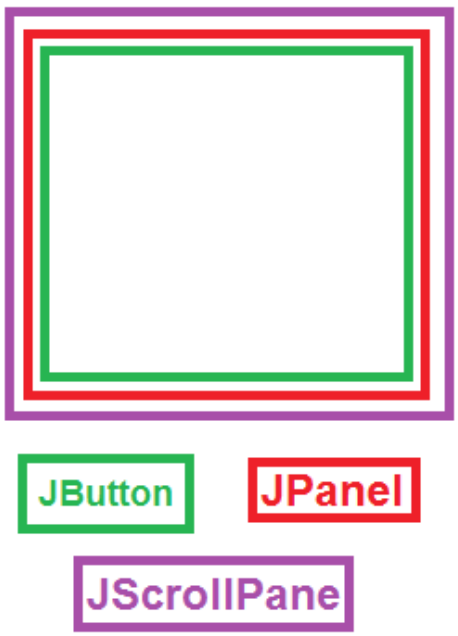


Illustration 4: Schema décrivantt les composants graphiques de l'affichage de la base de données

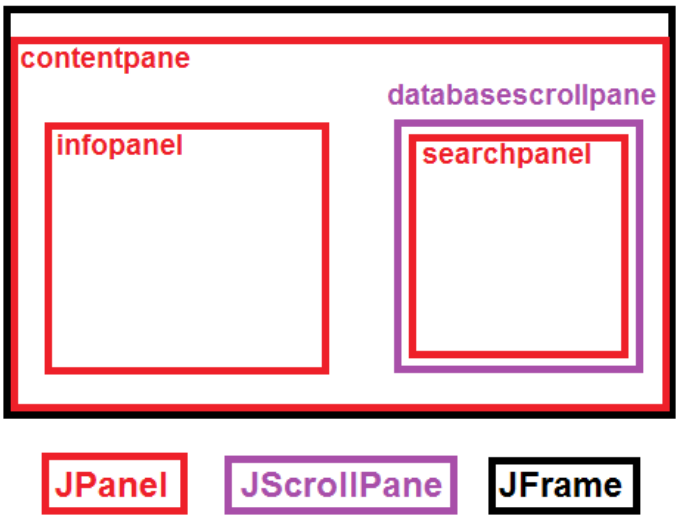


Illustration 5: Schema décrivant les différents containers de la fenêtre

Les actions

Chargement de fichiers

LoadFileAction: Fait appel à l'application JFileChooser pour charger des fichiers dans la base de données et seulement des fichiers car filechooser est appelé en mode FILES_ONLY. Définit des FileFilter pour pouvoir choisir de n'afficher que certaines extensions de fichiers. Fait ensuite appel à la procédure loadFile qui va charger le fichier sélectionné dans la base de donnée correspondante.

LoadDirectoryAction: Utilise aussi l'objet JFileChooser mais pour sélectionner un dossier et seulement un dossier car filechooser est définit en mode DIRECTORIES_ONLY. Fait appel à une procédure analogue à celle utilisée par le biais de la commande "-d" dans le CLI pour charger tous les .vcf/.ics du dossier.

Export de fichiers

ExportHtmlAction: Utilise JFileChooser en "mode save" pour pouvoir sélectionner la destination du fichier à exporter. Un FileFilter est aussi définit pour pouvoir choisir de n'afficher que les .html dans le dossier destination pour mieux voir si le fichier existe déjà. Un nom par défaut pour le fichier à exporter est ensuite proposé, et finalement, fait appel à la procédure validateExtension qui vérifie si l'extension du fichier à exporter est bien ".html" et la corrige au besoin.

ExportSerAction: Fonctionne de manière analogue à ExportHtmlAction mais avec à la place de ".html" une extension ".ser".

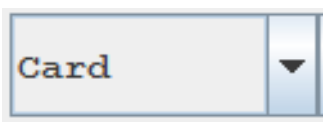
ExportObjectAction: De même, fonctionne de manière analogue à ExportHtmlAction et ExportSerAction mais avec soit ".vcf" soit ".ics" comme extension à la place de ".html" ou de ".ser".

Edition de fichiers

CreateNewObjectAction: Vide tous les champs de texte dans le panneau d'informations, définit l'objet sélectionné dans la base de donnée correspondante au format comme un objet vide.

SaveModificationsAction: Modifie les attibuts de l'objet sélectionné dans la base de donnée avec le contenu des zones de texte qui contiennent les informations correspondantes. Ajoute cet objet sélectionné à la base de données. Ce nouvel objet peut désormais être re-exporté, ou modifié au besoin.

Recherche et sélection du format



SelectFormatAction: Met à jour l'affichage de la base de données et change le contenu du panneau d'informations (les champs de texte dans la partie gauche) en fonction du format qui a été selectionné.

SearchAction: Ajoute tous les objets de la base de donnée (du format sélectionné) qui possèdent un ou plusieurs attributs correspondant à la recherche dans la liste searchresults. La recherche se fait par le biais des fonctions machingString définits dans chaque classe de données, ces fonctions retournent un booleen "vrai" si un de leurs attributs contient la donnée de la recherche. Enfin, l'affichage de la base de données est mis à jour pour montrer les résultats. Si rien n'est entré dans la recherche, la base de donnée entière est affichée. Cette action est aussi performée en appuyant sur Enter dans la barre de recherche.

Sélection d'un objet:

SelectObjectAction: Pour afficher les fichiers chargés dans la base de données, on ajoute à un JPanel (searchpanel) des JButton, à chacun de ces boutons on associe une action SelectObjectAction. Cette action prend en paramètre un indice (int) index, en résumé, à chaque bouton est associé une action et à cette action on associe un (int) index correspondant à l'indice de l'objet qu'il représente dans la liste database. De ce fait, chaque bouton est "lié" à un élément de la base de donnée par cet indice, on peut alors l'utiliser pour afficher le contenu de l'objet ainsi que pour remplacer l'attribut selectedcard/selectedcalendar par l'objet qui a été cliqué.

Ce principe est le même quand nous voulons sélectionner un objet qui est le résultat d'un recherche: nous associons à chaque bouton une action à laquelle est associé l'indice auquel se trouve l'objet sélectionné dans la liste searchresults. Puis l'attribut selectedcard/selectedcalendar est remplacé par le dernier objet sélectionné afin de pouvoir travailler avec (l'exporter, le modifier...).

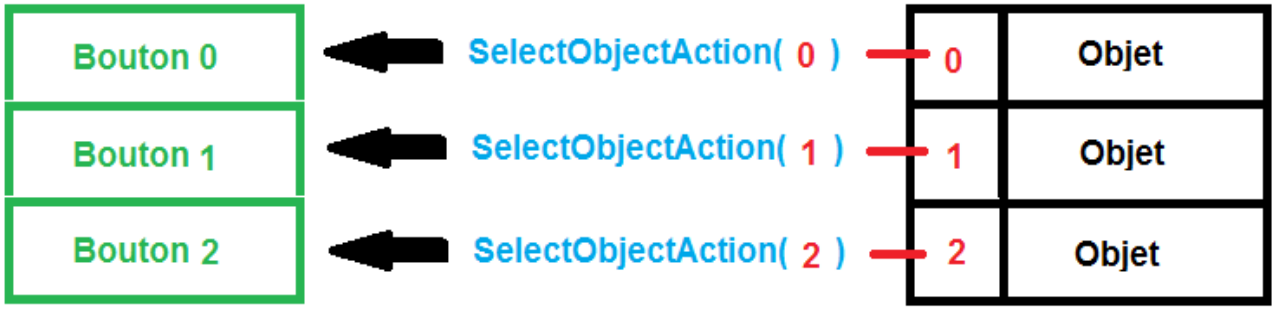


Illustration 6: Illustration de l'association des indices et actions aux boutons de sélection

7 Conclusion

Nous sommes plutôt heureux des résultats du développement des deux interfaces CLI et GUI. Dans l'aspect, ces deux programmes sont plutôt façiles à manier et bien aboutits. Cependant, à cause des délais à tenir pour rendre ce projet nous n'avons pas pu rendre, pour le GUI, un code qui utilise pleinement les connaissances acquises en cours. Par exemple nous aurions pu utiliser plus de polymorphisme pour traiter les objets Card et Calendar ou VDataGestion et IDataGestion. Nous ne l'avons pas fait car les développements des parties vCard et iCalendar se sont faits en décallé, il aurait donc fallut modifier tout le programme depuis le départ pour utiliser ce polymorphisme et nous n'aurions pas eu le temps. Cependant, dans le cas de la classe GUI.java par exemple, la gestion parallèle des deux formats vCard et iCalendar a facilité le développement du programme ainsi que le débogage. De plus, toujours dans le cas de la classe GUI.java nous aurions pu aussi diviser les

champs graphiques (affichage de la base de données, affichage des informations) en différentes classes pour désencombrer la classe GUI.java. Mais encore une fois tout avoir au même endroit a facilité le développement de l'application et nous n'avons pas eu le temps de modifier encore la structure car nous y avons pensé trop tard. Pour finir avec l'aspect technique, nous sommes certain que nous aurions pu insister plus sur la gestion d'exeptions mais nous n'avons rien trouvé qui soit réellement pertinent.

Enfin pour ce qui est de la répartition des tâches, nous nous sommes rendus compte au cours de nos cessions de travail que nous n'avions pas vraiment le même niveau en programmation, de ce fait la quantité de code fournit a été inégale dans le binôme. Nous nous sommes cependant assurés que nous savons mutuellement ce que l'autre a développé.