Projet de JAVA : Etude d'un document Geddoc

DARMON Cedric and DELAHOUSSE Matthieu

2 mai 2001

Chapitre 1

Realisation du projet

Nous avons passé quelques temps à étudier la hierarchie des classes pour la réalisation de ce projet. En effet la programmation objet et d'autant plus facile si la conception des classes avec leurs hiérarchies est bien effectuée.
Du point de vue de la programmation, celle ci c'est effectuée grace à la recherche de méthodes dans l'API et la création de methodes pour la réalisation du projet.
-> Le travail demandé :
L'objectif du projet consiste à écrire un programme en langage Java permettant de traiter une base de donnée généalogique constituée à partir d'un fichier au format GEDCOM. Le programme doit etre capable d'effectuer les operations suivantes :

- Lecture d'un fichier au format GEDCOM
- Traitement de la Cohérence
- La génèration de la documentation HTML

->Options Incluses :

- La génération d'un nouveau fichier Gedcom representant la base de donnée modifiée et corrigée
- Recherche par nom d'un individu
- Choix du placement des fichiers resultats
- Edition des individus manquants
- Page HTML d'un individu avec ses parents, ses freres et soeurs, ses conjoints et ses enfants
- Et quelques autres (Surtout pour les Incohérences).

1.1 Description d'un fichier Gedcom

la déscription du fichier Geddoc :

Les bases de données généalogiques sont maintenant organisées selon un standard largement adopté qui s'appelle GEDCOM. Voici une description des points essentiels pour le projet.

- Une généalogie est un graphe formé de deux espèces de sommets : les individus et les familles. Sur le dessin, les individus sont représentés par des rectangles horizontaux et l'unique famille par un rectangle vertical.

- Un sommet individu contient les données relatives à l'individu, en particulier son nom, ses prénoms, son sexe. D'autres informations peuvent être sa date et son lieu de naissance, de décès, etc. On peut lui associer des références de données multi-média (photos etc).
- Un sommet famille décrit une famille. Il contient peu d'informations (date du mariage s'il y a lieu, etc). Une famille est composée d'un père, d'une mère et d'un ensemble d'enfants.
- Les arcs du graphe vont des individus vers les familles, et des familles vers les individus (graphe bipartite).
- D'une famille partent trois types d'arcs : vers l'individu père, vers l'individu mère, et vers les individus enfants.
- D'un individu partent deux types d'arcs : un arc vers la famille où il figure comme enfant, et un arc vers chacune des familles où il figure comme parent. Un individu peut bien sûr apparaître dans plusieurs familles comme parent.

1.2 Le projet et sa hierarchie

Le projet consiste en la gestion d'une base de donnée comportant des individus et des familles. La génération de la documentation HTML demande une vérification de la cohérence de la base. Les différente étapes du programme sont donc dans l'ordre :

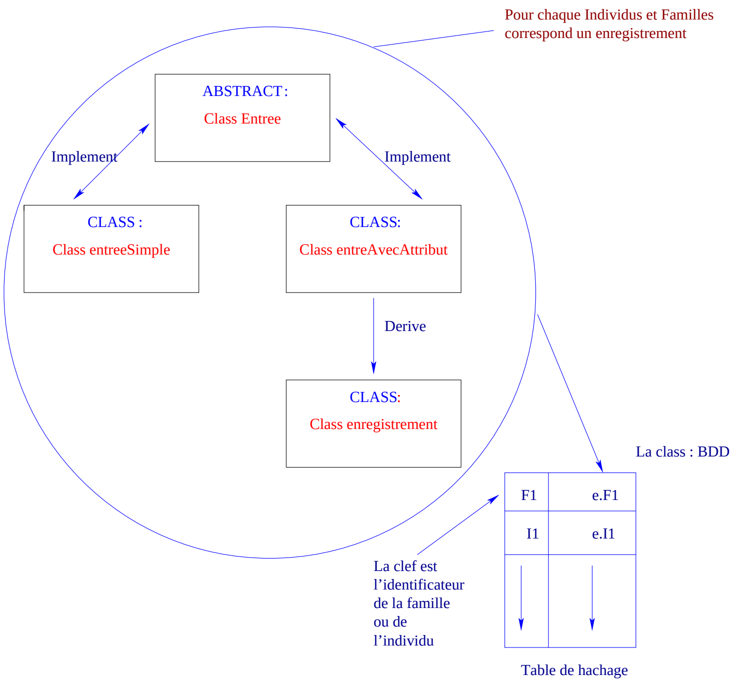
- lecture du fichier .ged
- constitution d'une base de données
- vérification de la cohérence de la base
- génération de la doc : une page par individu et par famille, index et résultat de recherche

Structures utilisées : La structure du fichier ged est completement conservée dans la structure de données : un fichier est composé d'enregistrements eux meme composés d' entrées simples - un type et une valeur - ou d'entrées avec attributs - un type et des attributs qui sont eux même des entrées simples et pourquoi pas des entrées avec attributs. On a donc un type abstrait `entree` : toute `entree` a un type. Des entrees sont soit des objects `entreeSimples` (une valeur en plus) soit des objects `entreeAvecAttributs` (un tableau attribut en plus). Un object `enregistrement` est un object `entreeAvecAttributs` particulier : il hérite de celui-ci mais possède un champ identificateur en plus. La base de données est composées d'objects `enregistrements` : le type d'un `enregistrement` est soit INDI soit FAM. Lors de la constitution de la base, on lit ligne par ligne le fichier ged. Le niveau nous renseigne sur le type d'entree :

- 0 on instancie un enregistrement : il a un type et un identificateur
- >0 on ajoute dans l'enregistrement courant soit l'`entreeSimple` soit l'`entreeAvecAttributs`

Nous reviendrons dans la section concernant la classe Analyse de la constitution des enregistrements.

Pour la compréhension de la hierarchie du projet, rien de mieux qu'un schéma permettant de représenter le lien entre les différentes class. Explication :.



1.3 ..Description des class..

Dans ce chapitre nous allons commenter les classes pour mieux comprendre leur utilisation. Il sera aussi utile pour la compréhension de regarder en même temps la doc java pour les méthodes de chaque class.

1.3.1 La classe entree

Nous avions besoin d'une classe abstraite pour regrouper les classes entreeSimple et enreeAttribuée. Elle permet de regrouper le String `type` et les méthodes :

- public String type : le type de l'entrée : FAM, IND mais aussi NAME, BIRT ; en fait on peut avoir n'importe quel "type"
- public static traduction traduc = new traduction() : table de hachage contenant les traductions des types ; static car commune à toute les entrées
- public abstract void afficheInfo() : affichage a l'écran des infos de l'entrée
- public abstract String toString() : affichage dans une chaine
- public abstract void vireValue(String types,String nouvel) throws CoherenceException : affichage dans une chaine
- public abstract String getValue(String types) throws CoherenceException : retourne la valeur du type
- public abstract void creerFichierGed(java.io.FileWriter fr,int niveau) : géneration dans un fichier .ged des informations sur l'entrée
- abstract void creerFichierHtml(java.io.FileWriter fr,Bdd genealogie) : géneration dans un fichier .html des informations sur l'entrée
- public boolean isA(String type) : l'entree est-elle du type passé en argu-

ment ?
Les méthodes abstraites nous obligent à les concretiser dans les classes entreeSimple et entreeAvecAttributs qui implémentent entree.

1.3.2 La classe entreeSimple

Comme nous l'avons expliqué, la classe entreeSimple représente un object composé d'un String type et d'un String valeur ; elle représente généralement les entrées de niveau 1 dans le fichier GedCom : 1 NAME JOHN BEGOOD est une entreeSimple de type NAME de valeur JOHN BEGOOD mais 2 DATE 22/05/1889 est aussi une entreeSimple.

1.3.3 La classe entreeAvecAttributs

Cette classe implémente aussi la classe entree. Elle ne possède pas de valeur comme entreeSimple mais a un tableau `attributs` ; la ligne 1 BIRT correspond à un object entreeAvecAttributs. Dans le tableau se trouvent les entrees de niveau inférieur au niveau de l'entreeAvecAttributs. On pourait donc avoir dans l'exemple un object entreeSimple de type DATE de valeur 22/05/1889 car dans le fichier ged son niveau serait 2 : on la stoke donc dans le tableau d'attributs de l'object entreeAvecAttributs de type BIRT. L'avantage de cette representation est de respecter le format du fichier puisque l'on conserve la hierarchie entre les données ; on vera que cela permet de définir reccursivement certaine fonctions et de pouvoir gerer facilement les enregistrements quand il faudra assurer la cohérence de la base. Deux méthodes interressantes : AjouteEntreeSimple(niveau,type,valeur) qui ajoute si le niveau est égale à 1 un nouvel object entreeSimple dans le tableau Attributs ; sinon, cherche le dernier object dans le tableau (qui doit etre une entreeAvecAttributs) et rappel la méthode sur cet object avec le niveau décrementé. L'autre méthode est similaire : AjouteEntreeAvecAttributs(niveau,type). L'avantage de ces 2 méthodes est que comme la lecture du fichier source se fait ligne par ligne, on regarde le niveau. S'il vaut 0, un nouvel enregistrment est crée ; pour les ligne de niveau supérieur, on lui applique ces 2 méthodes avec les Strings type et valeur lus.

1.3.4 La classe enregistrement

Au debut du projet nous avions décide de créer deux classes différentes : une classe pour les individus et une autre pour les familles. Cependant aprés reflexion, nous avons décidé de créer une seule classe étant donné la similitude des entrées. En effet, que se soit un individu ou une famille, les deux ont de toute façon des entrées d' attributs comme la date de mariage pour une famille ou bien le sex pour un individu. En fait, on ne s'interresse jamais aux valeurs des types : on pourait trés bien ajouter n'importe quel type dans un enregistrement. Cette abstraction permet de ne pas avoir à les gérer : on stoke toutes les informations à chaque fois. La classe implémente `Comparable` : ceci sera utile pour comparer des enregistrments selon la date de naissance ou le nom.

1.3.5 La génération des fichiers

Les classes entreeSimple et entreeAvecAttributs possèdent une métode pour écrire dans le fichier html : `creerFichierHtml`. Quand on veut obtenir la documentation d'un individu, on appel creerFichierHtml sur un object enregistrement. Cette métode créee physiquement un nouveau fichier identificateur.html et appel la métode creerFichierHtml pour chaque object entree contenue dans le tableau attribut. Si l'entree est une entreeAvecAttributs (par exemple BIRT) alors dans la métode creerFichierHtml on écrit "NAISSANCE" (pour être exacte la traduction fournie par la table de hachage de l'object traduction) puis on itére sur les entrees contenues dans le tableau attributs. Cette définition reccursive est utile aussi quand on veut obtenir un nouveau fichier .ged à partir d'une base que l'on a modifiée. Ceci est obtenu par la méthode `creerFichierGed` qui appelle recursivement sur les autres entrees et écrit ainsi de manière séquentielle dans le fichier les informations des entrees.

1.3.6 La classe Analyse

Dans un premier temps nous avons écrit la classe Analyse permettant comme son nom l'indique d'analyser le fichier Geddoc ligne par ligne avec un BufferedReader. Nous reconnaisons d'abord le debut du document (0 HEAD) et la fin du document (0 TRLR). Etant donné que les types d'entrée peuvent être "indéfinis" nous avons donc choisit de considerer deux Type de saisie : Le premier etant de niveau 0 et le deuxieme indéfini (1,2,3..). En effet une entrée de niveau 0 est la création d'un nouvel object enregistrement de type individu ou famille ; les autres entrées sont en quelque sorte les attributs de l'entrée de niveau 0 et ceci jusqu'a un nouvel enregistrement. Dans cette classe nous avons utilisé des stringTokenizers pour pouvoir recupperer les types, les identificateurs et les valeurs. Nous avons aussi decidé de ne pas reconnaitre les types comme NAME,SEX... en faisant de nombreux appels a la methode equals mais de les enregistrer en dur et ensuite de les traduire en français (ou en une autre langue). Ceci permet au programme de ne pas planter si un type n'est pas reconnut mais d'écrire celui ci dans le fichier html sans qu'il soit traduit.

1.3.7 La classe Interface

La classe Interface est assez indépendante des autres classes. Elle correspond tout simplement à l'appel des differentes classes pour l'execution des commandes. Dans un premier temps, elle est constituée de deux Textfield et de trois boutons correspondant à la fonction quitter (permet de quitter le programme avec confirmation), la fonction effacer (elle permet d'effacer les deux champs de texte) et enfin le bouton analyser qui correspond à l'analyse du document Geddoc pris en argument dans le premier champs de texte. Lors de la selection du bouton Analyser ,celui disparait et laisse place au bouton Check, qui permet de vérifier la cohérence de la base. Ensuite le bouton Chek disparait pour laisser place au bouton Création permettant de créer les pages HTML. Et pour finir nous avons les boutons Recherche et Index qui permettent respectivement d'effectuer la création de la page contenant les individus contenant un motif de recherche dans leur nom ou bien la page de tous les individus classés suivant le comparateur choisi. L'apparition et la disparition des boutons permettent un déroulement correct du programme sans que l'utilisateur n'est d'autre choix possibles.

1.3.8 La classe Bdd : Base de donnee

Pour stocker toutes les enregistrements, nous avons décidé de créer une table de hachache dont la clef est l'indentificateur d'un individu ou d'une famille et les objets sont des enregistrements. Ceci permet d'avoir les informations stockées et donc nous avoir accées a celle-ci lorsque nous avons besoin sans relire le fichier Geddoc. Etant donné que nous avons toutes les informations nécessaires, c'est dans cette classe que l'on vérifie la cohérence des données que nous avons enregistrées. Il est donc nécessaire pour vérifier la cohérence d'écrire de methode permettant la vérification des individus d'une part et des familles d'autre part.

verifieMembreFamille

- on itère sur les enregistrements de type FAM et on appel la méthode `verifieMembreFamille`
- on vérifie qu'il y a un seul père et une seule mère. Sinon resolutionSimple permet a l'utilisateur de choisir.
- on vérifie les membres : si un individu n'existe pas, `resolution` permet de le creer.
- on vérifie pour un pére par exemple qu' il a une entreeSimple de type SEX et de valeur M et une entreeSimple de type FAMS et de valeur l'identificateur de la famille considérée ; c'est le rôle de `verifieValeurTypeIndividu`. Quand il y a une absence, on ajoute une entreeSimple ; sinon, il faut une résolution manuelle quand il y a une contradiction ; c'est le role des classes resolutionSimple et resolution.
- On ajoute l'identificateur de l'individu dans le tableau `tousIndividus` ; ceci permettra de connaitre les individusSeuls, ceux qui ne sont pas référencés à travers les familles.

verifieIndividusSeuls

De la même manière, on vérifie l'unicité des types FAMC et SEX, de la présence des familles auquelles l'individu appartient : si elles sont absentes, il faut les créer. Si elles existent, il faut s'assurer qu'il n'y a pas de conflit comme la déclaration d'un parent : `resolutionSimple` permet de choisir

verifieCycle

On itère sur les individus. Ceux qui n'ont pas de type FAMC constiuent les `individusRacine`, ceux qui peuvent former un arbre ; en effet, on peut avoir un graphe non connexe. Pour un individusRacine, on calcul ses descendants. Un tableau `ancetre` memorise les ascendants. Il est donc au debut vide. Si un descendant appartient à ancetre, on a un cycle. Un tableau `visite` mémorise les individus visites : il faut y ajouter le la les conjoint(es) de celui ci car ils ont meme descendance.

1.3.9 La classe AgeComparator

Cette classe a était écrite pour pouvoir trier par age tous les individus de la base.Elle prend en paramettre deux objets que l'on cast ensuite en enregistrement pour pouvoir les comparer. C'est simplement un plus dans le programme pour pouvoir par exemple un index trié par age ou comme nous le verons par nom dans la classe NomComparator.

1.3.10 La classe NomComparator

Cette classe et tous simplement la même que celle pour les ages cepenedant celle ci concerne les noms.On pourra ainsi obtenir un index de recherche ou d'individus triée par nom en ordre croissant

1.3.11 La classe CoherenceException

Nous avons défini dans ce projet une classe CoherenceException qui permet de lever nos propres exceptions en l'occurence ici lorsque l'on est confronté a une erreur de coherence dans le fichier Geddoc.Si cette exception est levée le programme ne peut plus continuer,et signal à l'utilisateur que l'exception est levée.L'erreur ne pourra donc pas être corrigée par les méthode de la classe Bdd.

1.3.12 La classe Resolution

Lorsque nous relevons des contradictions dans la base, il faut pouvoir proposer à l'utilisateur une résolution. Cette classe donne donc le choix a l'utilisateur de modifier la base de données. Exemple de résolutions :

- `ajout et édition d'un individu référencé dans une famille mais absent` :
  Nous demandons alors a l'utilisateur de saisir une carte d'identité pour l'individu en question. Cette carte d'identité est constituée des informations les plus importantes car nous n'avons pas voulu surcharger la carte d'identité d'un individu.
- `résolution des doublons`
  Les données rentrées par l'utilisateur seront aussi verifiées pour ne pas rajouter des nouvelles erreurs dans la base. Si un individu existe deux fois dans le fichier Gedcom nous demandons à l'utilisateur de choisir un enregistrement entre les deux enreistrements concernant le meme individu.

1.3.13 La classe ResolutionSimple

cette classe complète la classe Resolution ; en effet elle resoud les contradictions. Ces résolutions de type simple sont :

- `résolution des doublons dans une meme famille` :
  Lorsque nous retrouvons dans la base un des deux parents qui se retrouvent enfants dans une meme famille. Ce type d'incohérence n'est pas résolu ,en effet nous ne savons pas ou est l'erreur dans la base.Nous demandons donc à l'utilisateur de verifier le fichier Gedcom.
- `individu ayant un type sex non-conforme dans une famille` :
  La deuxieme résolution est de type SEX ; en effet lorsque un individu n'a pas de sex nous l'ajoutons dans la base si il n'est pas nécessaire de

connaitre son sex. Cependant lorsque l'individu appartient à une famille nous demandons a l'utilisateur si il veut l'inserer en tant que père et homme ou mère et femme. La modification peut entrainer d'autres résolutions .

- `individu ayant un type sex absent pour un individu` :
  Si un individu est parent d'une famille mais qu'on ne sait pas si il est pére ou mère, la resolution permet de spécifier son sex pour deux chose : -ajouter son sex M ou F a un individu. -et ajouter son statut dans la famille.
- `individu appatenant à le meme famille plusieurs fois` :
  Dans ce cas, nous ne savons pas résoudre cette incohérence et nous demandons a l'utilisateur de vérifier le fichier Gedcom. En effet nous ne pouvons pas savoir si l'individu est pére ou mére ou enfant.
- `Si un individu appartient à plusieurs familles en tant que FAMC`:
  Si un individu appartient à plusieurs famille nous demandons à l'utilisateur dans quelle famille il veut l'insérer, et nous l'inserons dans sa bonne position.
- `Si dans une famille nous avons plusieurs pére “et ou” mère` :
  Nous demandons à l'utilisateur de choisir l'individu correspondant au type demandé (soit mère soit pére).Ensuite nous supprimons toutes les entreeSimples de type WIFE ou HUSB en nous rajoutons les entrées selectionnées.

1.3.14 La classe Traduction

C'est une table de hachage construite à partir d'un fichier traduction contenant les couple (TYPE TRADUCTION) Ceci permet d'obtenir des pages HTML compréensible et d'ojouter facilement des types qui seraient absents dans le standard.

1.4 Conclusion et développement

Certaines Incohérences du fichier Gedcom ne peuvent pas etre résolues étant donné que nous ne connaissons pas la réponse a cette incohérences. Cependant nous avons traité toutes les inchoérences demandées dans le sujet ainsi que certaines que nous avons rajoutés. Grace a la classe Traduction nous pourrions aussi proposer à l'utilisateur une traduction des Type de base (ou référencé) en plusieur langues. Un autre point pouvant etre amelioré consisterait à utiliser une table de hachage au lieu d'un tableau d'attributs (pour une entreeAvecAttribut ou un enregistrement) car ceci aurait permis la suppression des nombreux iterators. Cependant ceci pose des problemes : en effet on aurait du changer la structure des méthodes AjouteentreeSimple et AjouteentreeAttribuee.

Table des matières