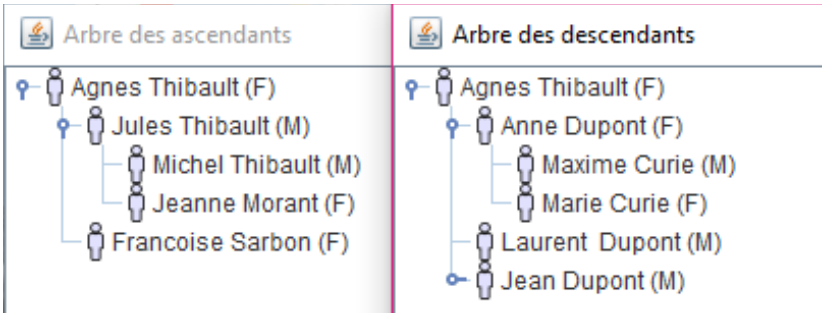
ENSG Géomatique
ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

IGN
INSTITUT NATIONAL DE L'INFORMATION GÉOGRAPHIQUE ET FORESTIÈRE

DOCUMENT DE SYNTHÈSE
PROJET INFORMATIQUE

« Gestion d'un Arbre Généalogique »



Anouk Schleich
Marie Stretti

ENSG - ING1
Mai 2018

I. Bilan du projet

1. Organisation

Nous avons travaillé en binôme pour l'analyse du sujet et pour la réalisation des différents diagrammes UML (cf Rapport d'Analyse), afin de bien définir le projet, les limites et les étapes.

Pour l'implémentation du code, nous avons commencé par travailler ensemble, puis nous nous sommes réparties les différentes classes et méthodes à implémenter. Lorsque l'une de nous deux rencontrait des difficultés alors nous regardions à deux pour résoudre le problème.

La rédaction des rendus demandés a été réalisée en binôme.

2. Travail réalisé

Au bout de ces six semaines nous avons réussi à coder les attributs et les relations nécessaires à l'élaboration d'un arbre généalogique. L'utilisateur peut ajouter des personnes à l'arbre et les modifier, et ajouter et modifier des attributs et des relations entre celles-ci. De plus, nous avons implémenté l'affichage d'une fenêtre avec l'arbre des descendants et des ascendants d'une personne. Une fiche identité s'affiche lorsqu'on double-clique sur une personne dans l'arbre.

3. Limites

Il nous manque quelques contrôles lors de l'ajout ou la modification d'attribut(s) et de relations. Par exemple, il faudrait vérifier que les dates (naissance, décès, mariage…) rentrées soit bien antérieures à la date actuelle. Il faudrait également vérifier que la chaîne de caractères entrée pour une date corresponde réellement à une date et donc interdire un format différent du format ‘'AAAA-MM-JJ''. De plus, tous nos tests de cohérence se limitent à l'année, nous ne vérifions ni le mois ni le jour.

Pour Naissance et Décès nous avons fait une seule source commune aux deux attributs date et lieu. Ainsi, lorsqu'on souhaite modifier soit la date soit le lieu il faut obligatoirement entrer l'autre aussi, ce qui n'est pas optimal pour l'utilisateur. Ceci serait donc à améliorer.

Lorsqu'un mariage est créé, il faut le garder dans le main, sinon on ne peut plus ajouter de témoins. Il aurait fallu faire une méthode pour rendre possible l'ajout de témoin(s) à un mariage qui n'est plus dans le main. Nous n'avons pas ajouté le nom du mari au nom de jeune fille de la mariée, précédé de la mention ep.. Cela nous semblait peu important et difficile à gérer lorsque la personne s'est mariée plusieurs fois.

L'interface peut être améliorée. Notre projet nécessite d'appeler les méthodes par le main pour modifier ou ajouter des attributs et relations. Il serait préférable de tout entrer dans la console, ou même dans une interface graphique. Ceci éviterait à l'utilisateur d'accéder au code java et faciliterait la modification et l'ajout de données.

Notre application et nos méthodes font toujours appel à la BDD. Ceci est très répétitif et

nous aurions pu éviter certains de ces appels en créant les attributs dans notre classe Personne.

4. Apprentissage

Nous avons revu les différents types de schéma UML et utilisé pour la première fois des logiciels pour la création de nos diagrammes : Dia et StarUML.

Nous avons approfondi nos connaissances en programmation orienté objet sous Java et découvert une nouvelle SGBD : SQLite.

Nous nous sommes rendues compte de l'utilité d'afficher des phrases temporaires le temps de tester une méthode pour mieux comprendre où étaient les erreurs. De plus, nous avons vu qu'il est important de choisir des noms pertinents pour les attributs et méthodes pour se retrouver ensuite, notamment lorsqu'on travaille en groupe et pour comprendre le code écrit par l'autre. Nous avons mieux compris la différence entre un attribut/une méthode statique et non statique, différence qui restait floue avant.

II. Développement

1. Organisation des développements

Le code est composé de 9 classes (**Personne**, **Metier**, **Mariage_mairie**, **Mariage_eglise**, **Bapteme**, **Sepulture**, **Tree**, **Connexion** et **Main**) et de 2 énumérations (**Sexe** et **Source**). Au départ nous avions créé une classe **Acte_religieux** dont **Bapteme**, **Sepulture** et **Mariage_eglise** héritaient. Finalement, nous l'avons supprimée car les trois classes ont des méthodes trop différentes pour avoir une même classe mère.

La plupart des méthodes s'appliquent à une personne et sont donc implémentées dans la classe **Personne**. C'est dans cette classe que l'on ajoute, modifie et supprime les attributs d'une personne (prénom, nom, sexe, date de naissance, lieu de naissance, date de décès, lieu de décès , la source des informations de naissance et la source des informations de décès). On ajoute, modifie et supprime également les relations frère/sœur, enfant/parent, conjoint, job, baptême et sépulture.

Les méthodes *add* ajoutent les attributs de la personne dans la BDD et les relations entre deux personnes. Ces méthodes se connectent à la BDD et ajoutent la relation si elle n'existe pas déjà.

Lorsque deux personnes ont un même parent ils sont automatiquement ajoutés en tant que frères avec la méthode *ajoutFrere(Personne parent)*.

La méthode *addJob* ajoute le métier entré en paramètre à une personne. Nous pensions au départ faire une énumération de métiers mais nous nous sommes vite rendues compte que ce n'était pas optimal puisque cela obligeait à rentrer tous les métiers d'un seul coup alors qu'ils n'étaient pas encore connus. De plus, nous aurions récupéré les identifiants des métiers et non les noms, ce qui n'était pas parlant.

Pour les événements mariage, sépulture, et baptême nous avons décidé de relier un événement à une personne en deux étapes. Dans un premier temps, on crée l'événement avec ses attributs en utilisant le constructeur dans sa classe respective (**mariage_eglise, mariage_mairie, bapteme, sepulture**) puis on ajoute ces événements à une personne dans la classe **Personne**.

ex.: Créer une sépulture et l'ajouter à une personne

- créer une sépulture
  Sepulture sep = **new** Sepulture ("2016-22-09", "Paris", Source.acte_officiel);
- ajouter la sépulture à une personne
  pers1.addSepulture(sep);

Pour ajouter un témoin, il est nécessaire d'avoir créé un mariage.

Les méthodes *change* sont utilisées pour les relations et événements uniques. On peut modifier le prénom, le nom, la date de naissance, le lieu de naissance, la date de décès, le lieu de décès, la sépulture et le baptême associés à la personne. Le conjoint d'une personne est unique : si nous le modifions, la méthode supprime l'ancien conjoint et le remplace par le nouveau. De plus, lorsque nous changeons un événement, la méthode vérifie si un événement du même type est déjà attribué à la personne. Si ce n'est pas le cas, il ne peut pas être modifié et un message d'avertissement s'affiche dans la console.

Pour modifier les données de naissance, décès, baptême et sépulture il faut indiquer la source de l'information. La méthode compare ensuite la source des anciennes informations avec la nouvelle source pour déterminer si elle met à jour ou non les attributs.

Les méthodes *coherence* contrôlent les âges des personnes ainsi que les différentes dates utilisées dans les méthodes. Par exemple, pour le baptême on vérifie que celui-ci a lieu après la date de naissance de l'enfant et pour la sépulture on vérifie que la date de la sépulture soit postérieure à la date de décès.

L'affichage de l'arbre généalogique nous a pris du temps, car nous avons exploré de nombreuses possibilités d'affichage avant de trouver le Jtree. Nous avons perdu beaucoup de temps à essayer de comprendre des codes de forum et de nous les approprier. Nous avons finalement deux méthodes d'affichage : *ArbreDescendance()* et *ArbreAscendance()*. On visualise dans une fenêtre les enfants de la personne, puis les petits-enfants, puis les enfants des petits-enfants, etc pour l'arbre descendant. De même pour l'arbre ascendant qui montre donc les parents de la personne, puis les grand-parents puis les parents des grands-parents, etc. La fiche identité utilise toutes les méthodes *get*.

2. Exécuter le programme

Tout d'abord il faut vérifier que SQLite soit installé sur le poste et le cas échéant le télécharger.

Ensuite il y a deux possibilités :

- exécuter directement le programme pour visualiser l'arbre et les fiches identités
- entrer des commandes dans le main pour modifier/ajouter des attributs et/ou relations

Annexes – Journal de bord

mardi 03/04 :

- Analyse du sujet : limites et compréhension
- Début du diagramme de cas d'utilisation

lundi 09/04 : finition des 3 diagrammes

- Diagramme de classes : Nous avons choisi de mettre la relation *est_temoin* entre les classes **mariage_eglise** et **personne** et non de faire une association réflexive (cas compliqué quand plusieurs mariages). Ceci permet d'avoir plusieurs témoins pour un seul mariage et une personne peut être témoin de plusieurs mariages. De même pour la relation *est_parrain/marraine* entre les classes **bapteme** et **personne**.
- Création de la classe abstraite ***acte_religieux*** et des trois classes en héritant, **mariage_eglise**, **sepulture**, **bapteme**. En effet pour les trois cas on a les mêmes attributs.
- Diagramme de cas d'utilisation : séparation des relations et attributs. Nous avions mis les deux ensemble mais finalement ce sont deux cas très différents. Pour une relation il faut deux personnes, ou une personne et un mariage, alors que pour un attribut il faut une personne et il faut entrer une donnée.
- Diagramme d'activités : séparation des relations et attributs
- Début de rédaction du rapport d'analyse

mardi 10/04 :

- Création de la BDD sur BD Browser for SQLite à partir du diagramme de classes : Nous avons choisi de faire des tables pour les relations. Il existe donc une table **parent**, **frere_sœur**, **conjoint**, **temoin_mairie** et **temoin_eglise**. Ces tables ont chacune deux colonnes qui renvoient aux identifiants de **personne** ou éventuellement à l'identifiant du mariage.
  Nous n'avons pas créé de table pour parrain/marraine car une personne baptisée possède un et un seul parrain. Ce dernier est donc représenté dans la classe **bapteme** par une clé étrangère.
- Remplissage de la BDD avec BD Browser
- Création des classes dans Eclipse
- Connexion à SQLite avec java
- Création de la BDD relationnelle
- Suite de la rédaction du rapport

vendredi 20/04 :

- Programmation de la classe **frere**
- Programmation de la classe **parent**
- Suite de la rédaction du rapport

mercredi 25/04 :

- Modification des classes **parent**, et **frere**
- Suite de la rédaction du rapport

jeudi 26/04 :

- Implémentation des classes **conjoint** et **job**

vendredi 27/04 :

- Suite de la rédaction du rapport

jeudi 03/05 :

- Tentative d'import de la table personne dans une liste de la classe **Personne**
- Tentative de suppression des doublons dans les relations

vendredi 04/05 :

- Suppression des classes **Parent**, **Frere** et **Conjoint**
- Création des méthodes *addParent*, *addEnfant*, *addFrere*, *addConjoint*, *addJob* dans la classe **Personne** avec gestion des doublons
- Importation de la table personne de la BDD dans une liste (listePers) de la classe **Personne**, les colonnes de la table correspondent aux attributs de la classe
- Instanciation d'une personne
- Finalisation du rapport d'analyse

lundi 07/05 :

- Ajout de la Javadoc et de commentaires dans le code

mardi 08/05:

- Création des méthodes *mariages*, *addBapteme*, *addSepulture*
- Programmation de la relation Parrain
- Programmation de la déclaration des témoins pour les mariages

mercredi 09/05

- Création des méthodes *changeNaissance*, *changeDeces* pour changer les attributs d'une personne en prenant en compte la source

jeudi 10/05 :

- Création des méthodes *changeBapteme*, *changeSepulture* pour changer les attributs d'une personne en prenant en compte la source

vendredi 10/05 :

- Recherche internet sur l'affichage d'un arbre
- Essai vain de coder un arbre

dimanche 13/05 :

- Début de rédaction rapport de synthèse

lundi 14/05 :

- Suppression de la classe abstraite ***Acte_religieux***, car non utilisée par les relations
- Création de méthodes *get* pour afficher les relations
- Programmation d'une classe **ArbreAscendant** utilisant un Jtree avec une méthode *ajoutNoeud*
- Programmation d'une classe **ArbreDescendant** utilisant un Jtree
- Création des méthodes *trouveEnfant* et *trouveParent*, qui renvoient une liste de Personne

mardi 15/05 :

- Les deux classes d'arbre étant très ressemblants, recodage pour n'avoir qu'une seule classe **Tree** qui peut faire un arbre des descendants et des ascendants avec

une seule méthode.
- Amélioration des méthodes *get* des relations
- Implémentation des test de cohérence entre âge Parent/Enfant , âge entre conjoints
- Implémentation de tests d'obligation Témoin > 18 ans le jour du mariage, enfant plus jeune que son parent, sépulture après date de décès, baptême après date de naissance et vérification que la date de décès soit bien postérieure à la date de naissance

Mercredi 16/05 :
- Rédaction du rapport de synthèse
- Implémentation dans **Tree** pour afficher une fiche caractéristique en cliquant sur une personne dans l'arbre généalogique avec la méthode *doMouseClicked*
- Amélioration des méthodes *get* des relations
- Rédaction du ReadMe

jeudi 17/05 :
- Finalisation du rapport de synthèse
- Test final du programme
- Début préparation pour la soutenance
- Rendu du projet sur GitHub