

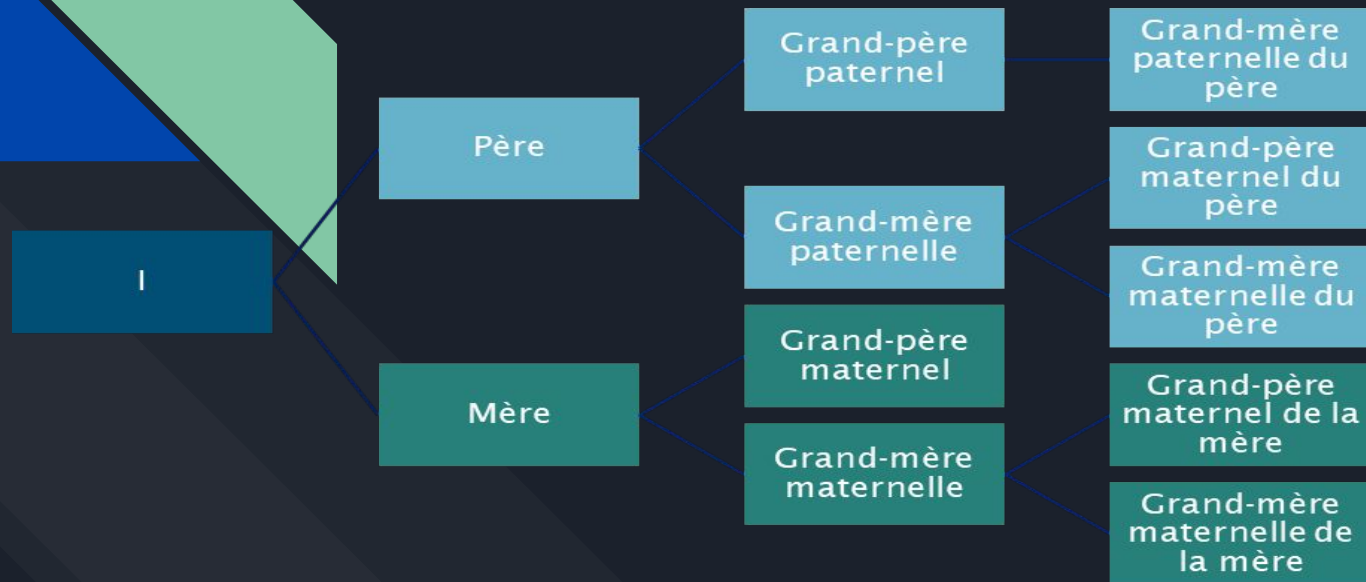
UE Projet : Arbre généalogique

ING 1 GM Groupe 8

*Salim OUESLATI
Bacarie TCHABO
Mehdi TORJMEN
Roshanth RUBAN
Iyad BEN MOSBAH
Kathirvele RANGANADANE*

*Encadrante
Zaouche DJAOUIDA
CY Tech / 2023-2024*

Présentation





I. Introduction

A - Description des variables

Variables quantitatives

Discrètes

- Identifiant unique
- Identifiant du père
- Identifiant de la mère

Variables qualitatives

Nominales

- Le nom
- Le prénom
- La nationalité

Dichotomiques

- Présence d'antécédents de diabète
- Présence du diabète (Outcome)



I. Introduction

B - Manipulation du fichier familles.csv

- Vérification de la cohérence du fichier
- Suppression une ligne du fichier
- Ajout d'une ligne au fichier



I. Introduction

C - Consultation de l'arbre généalogique global

Visualiser :

- L'arbre généalogique global
- Une partie de l'arbre généalogique relatif à une famille donnée
- La descendance d'une personne donnée
- L'ascendance d'une personne donnée
- Les frères, sœurs ou autres proches d'une personne donnée
- La liste des personnes sans ascendance



I. Introduction

D - Prédiction des risques de survenue d'un diabète

Prédictions doivent se baser sur ces facteurs :

- antécédents de diabète
- types de diabète

Méthode de prédictions :

- une méthode de clustering comme K-means (python)

E - Interface graphique

- visualiser et modifier facilement l'arbre généalogique (java)



II. Base de données : Règle de cohérence

Liste des attributs :

ID Personne

ID Père, ID Mère,

ID Conjoint

Nom, Prénom, Date Naissance,
Nationalité, Sexe

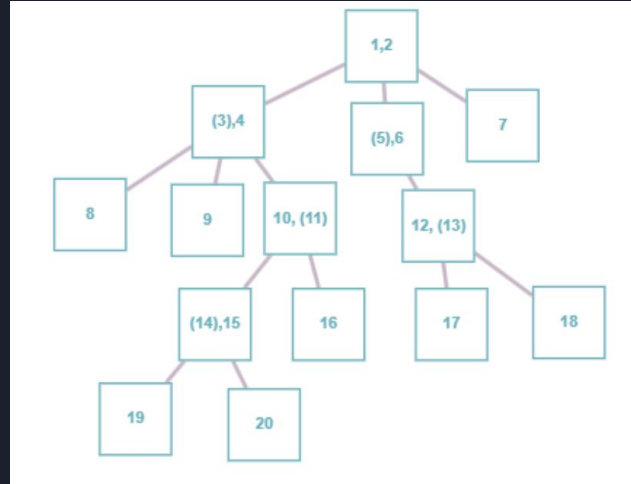
IMC ,Taux d'insuline ,Taux de
glucose, Outcome

Règle de cohérence :

1. L'unicité des identifiants
2. Vérifiez la cohérence et le format de la date de naissance
3. L'existence de parents
4. Pas de relation incohérente
5. L'absence de cycles dans les relations

Construction de la base de données

Le fichier familles.csv est une base de données comportant les informations d'individus issus de différentes familles.



Graphe d'aide à la construction

`"4", "Dubois", "Hugo", "1995-11-25", "FR", "2", "1", "3"`

Format de la base de données



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Personne : Attributs

Niveaux d'accès	Type	Identificateur	Description
private	String	id	Identifiant de la personne
private	String	nom	Nom de la personne
private	String	prenom	Prénom de la personne
private	String	dateNaissance	Date de naissance de la personne
private	String	nationalite	Nationalité de la personne



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Personne : Attributs (2)

Niveaux d'accès	Type	Identificateur	Description
private	String	idPere	Identifiant du père de la personne
private	String	idMere	Identifiant de la mère de la personne
private	String	idConjoint	Identifiant du conjoint de la personne



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Personne : Attributs (3)

Niveaux d'accès	Type	Identificateur	Description
private	Liste de Personne	conjoint	Liste des conjoints de la personne
private	Liste de Personne	enfants	Liste des enfants de la personne



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Personne : Méthodes

Les méthodes sont simplement les getters (accesseurs) et setters (mutateurs) des différents attributs.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Attributs

Niveaux d'accès	Type	Identificateur	Description
private	Map d'Integer à Personne	personnesMap	Map permettant de stocker les personnes par leur identifiant
private	Liste de Personne	roots	Liste des racines de l'arbre généalogique



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes

getPersonnesMap :

- Niveau d'accès : public
- Type de retour : Map<Integer, Personne>
- Description : il s'agit du getter (accesseur) de la variable personnesMap

getRoots :

- Niveau d'accès : public
- Type de retour : List<Personne>
- Description : il s'agit du getter (accesseur) de la variable roots



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (2)

readData :

- Niveau d'accès : public
- Type de retour : void
- Paramètres : variable pathname de type String
- Description : cette méthode permet de lire depuis un fichier CSV et remplit la map des personnes et la liste des racines.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (3)

Description détaillée de la méthode **readData** :

- Déclaration de la variable **personne** : une variable **personne** de type **Personne** est déclarée et initialisée à **null** et servira à stocker temporairement chaque **Personne** lue depuis le fichier CSV.
- Bloc **try** : on commence un bloc **try** pour capturer et gérer les exceptions qui pourraient survenir lors de la lecture du fichier.
- Ouverture du fichier CSV :
 - On crée un fichier **FileReader** pour lire le fichier situé à **pathname**.
 - On utilise un **CSVReader** de la bibliothèque **opencsv** pour lire les données CSV. **CSVReader** est spécialisé dans la lecture de fichiers CSV et gère correctement les délimiteurs et les citations.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (4)

Description détaillée de la méthode **readData** (2) :

- Déclaration de **nextLine** : une variable de type **String[]** est déclarée pour stocker chaque ligne lue du fichier CSV.
- Boucle **while** : lit chaque ligne du fichier CSV jusqu'à ce qu'il n'y ait plus de lignes à lire (**csvreader.readNext()** retourne **null** lorsqu'il atteint la fin du fichier).
 - **nextLine = csvreader.readNext()** : lit la prochaine ligne du fichier CSV.
 - **String[] data = nextLine** : stocke la ligne lue dans le tableau **data**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (5)

Description détaillée de la méthode **readData** (3) :

- Création d'une **Personne** : instancie un objet **Personne** en utilisant les données lues. Les indices du tableau **data** correspondent aux différents champs de la classe **Personne** (par exemple, **data[0]** pourrait être l'ID, **data[1]** le nom, etc.).
- Ajout aux racines : vérifie si la personne lue n'a pas de père (**getIdPere().equals("0")**) ou de mère (**getIdMere().equals("0")**). Si c'est le cas, elle est ajoutée à la liste des **roots** qui stocke les personnes sans ascendants connus, donc potentiellement des racines de l'arbre généalogique.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (6)

Description détaillée de la méthode **readData** (4) :

- Ajout à la map **personnesMap** : ajoute la **Personne** à la map **personnesMap** où la clé est l'ID de la personne (convertie en entier avec **Integer.valueOf(personne.getId())**) et la valeur est **Personne**.
- Fermeture du lecteur CSV : **csvreader.close()** : ferme le **CSVReader** pour libérer les ressources associées au fichier.
- Bloc **catch** : si une exception survient pendant la lecture ou le traitement du fichier, elle est capturée et son message d'erreur est imprimé avec **e.printStackTrace()**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (7)

Description détaillée de la méthode **readData** (5) :

En résumé, la méthode **readData** lit un fichier CSV ligne par ligne, crée des objets **Personne** à partir des données lues, les ajoute à une map **personnesMap** pour un accès rapide par ID et à une liste **roots** si la personne n'a pas de père ou de mère, indiquant qu'elle est une racine de l'arbre généalogique.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (8)

getEnfantsById :

- Niveau d'accès : private
- Type de retour : List<Personne>
- Paramètres : variable id de type String
- Description : cette méthode retourne la liste des enfants d'une personne à partir de son identifiant.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (9)

Description détaillée de la méthode **getEnfantsById** :

- Déclaration de la liste **enfants** : crée une nouvelle liste **ArrayList** de type **Personne** pour stocker les enfants trouvés.
- Boucle **for** : itère sur chaque **Personne** dans la collection des valeurs de la map **personnesMap**. La map **personnesMap** contient toutes les personnes avec leur identifiant comme clé.
- Condition pour vérifier l'identifiant du père (**if (p.getIdPere().equals(id))**) : vérifie si l'identifiant du père de la personne **p** est égal à **id**, l'identifiant passé en paramètre.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (10)

Description détaillée de la méthode **getEnfantsById** (2) :

- Ajout à la liste **enfants** (**enfants.add(p)**) : si la condition est vraie (c'est-à-dire si **p** a **id** comme père), ajoute la personne **p** à la liste **enfants**.
- Fin de la boucle **for** : la boucle continue d'itérer sur toutes les personnes dans **personnesMap**.
- Retour de la liste **enfants** : après avoir vérifié toutes les personnes, retourne la liste **enfants** qui contient toutes les personnes ayant **id** comme identifiant de père.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (11)

Description détaillée de la méthode **getEnfantsById** (3) :

En résumé, la méthode **getEnfantsById** parcourt toutes les personnes stockées dans **personnesMap**, vérifie si l'identifiant du père de chaque personne correspond à **id** et si c'est le cas, ajoute cette personne à une liste **enfants**. À la fin, la méthode retourne cette liste, fournissant ainsi tous les enfants du père dont l'identifiant est **id**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (12)

getConjoints :

- Niveau d'accès : private
- Type de retour : List<Personne>
- Paramètres : variable id de type String
- Description : cette méthode retourne la liste des conjoints d'une personne à partir de son identifiant.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (13)

Description détaillée de la méthode **getConjoints** :

- Déclaration de la liste **conjoints** : crée une nouvelle liste **ArrayList** de type **Personne** pour stocker les conjoints trouvés.
- Boucle **for** : itère sur chaque **Personne** dans la collection des valeurs de la map **personnesMap**. La map **personnesMap** contient toutes les personnes avec leur identifiant comme clé.
- Condition pour vérifier l'identifiant du conjoint (**if (p.getIdConjoint().equals(id))**) : vérifie si l'identifiant du conjoint de la personne **p** est égal à **id**, l'identifiant passé en paramètre.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (14)

Description détaillée de la méthode **getConjoints** (2) :

- Ajout à la liste **conjoints** (**conjoints.add(p)**) : si la condition est vraie (c'est-à-dire si **p** a **id** comme père), ajoute la personne **p** à la liste **conjoints**.
- Fin de la boucle **for** : la boucle continue d'itérer sur toutes les personnes dans **personnesMap**.
- Retour de la liste **conjoints** : après avoir vérifié toutes les personnes, retourne la liste **conjoints** qui contient toutes les personnes ayant **id** comme identifiant de conjoint.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (15)

Description détaillée de la méthode **getConjoints** (3) :

En résumé, la méthode **getConjoints** parcourt toutes les personnes stockées dans **personnesMap**, vérifie si l'identifiant du conjoint de chaque personne correspond à **id** et si c'est le cas, ajoute cette personne à une liste **conjoints**. À la fin, la méthode retourne cette liste, fournissant ainsi tous les conjoints de la personne dont l'identifiant est **id**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (17)

buildHierarchyTree :

- Niveau d'accès : public
- Type de retour : void
- Paramètres : variable root de type Personne
- Description : cette méthode construit l'arbre de hiérarchie en ajoutant récursivement les enfants.




III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (18)

Description détaillée de la méthode **buildHierarchyTree** :

- Initialisation de la variable **personne** : assigne la valeur **root** à la variable **personne**. Cela simplifie les références à **root** dans le reste de la méthode.
- Obtention des enfants de la racine (**List<Personne> enfants = getEnfantsByld(personne.getId())**) : appelle la méthode **getEnfantsByld** avec l'identifiant de **personne** pour obtenir une liste de ses enfants et l'assigne à la variable **enfants**.
- Obtention des conjoints de la racine (**List<Personne> conjoints = getConjointsByld(personne.getId())**) : appelle la méthode **getConjoints** avec l'identifiant de **personne** pour obtenir une liste de ses conjoints et l'assigne à la variable **conjoints**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (19)

Description détaillée de la méthode **buildHierarchyTree** (2) :

- Affectation des enfants et conjoints à la racine :
 - **personne.setEnfants(enfants)** : assigne la liste **enfants** à l'attribut **enfants** de **personne**.
 - **personne.setConjoints(conjoints)** : assigne la liste **conjoints** à l'attribut **conjoints** de **personne**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (20)

Description détaillée de la méthode **buildHierarchyTree** (3) :

- Vérification des enfants :
 - **if(personne.getEnfants().isEmpty())** : vérifie si la liste des enfants de **personne** est vide.
 - **return** : si **personne** n'a pas d'enfants, la méthode retourne immédiatement, car il n'y a pas d'autres descendants à traiter.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (21)

Description détaillée de la méthode **buildHierarchyTree** (4) :

- Construction récursive de l'arbre :
 - **for(Personne p : personne.getEnfants())** : itère sur chaque enfant de **personne**
 - **buildHierarchyTree(p)** : appelle récursivement **buildHierarchyTree** sur chaque enfant **p**, construisant ainsi le sous-arbre pour chaque enfant.




III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (22)

Description détaillée de la méthode **buildHierarchyTree** (5) :

En résumé, la méthode **buildHierarchyTree** construit l'arbre généalogique de manière récursive en partant d'une **Personne** donnée (la racine). Elle récupère les enfants et conjoints de cette personne, les assigne à la personne, puis appelle récursivement la méthode pour chaque enfant, construisant ainsi l'arbre descendant complet. Si la personne n'a pas d'enfants, la méthode retourne immédiatement, terminant ainsi la construction pour cette branche de l'arbre.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (23)

printHierarchyTree :

- Niveau d'accès : public
- Type de retour : void
- Paramètres :
 - Variable root de type Personne
 - Variable level de type int
- Description : cette méthode affiche l'arbre de hiérarchie à partir de la racine et du niveau initial.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (24)

Description détaillée de la méthode **printHierarchyTree** :

- Indentation pour afficher le niveau :
 - **for(int i = 0; i < level; i++)** : une boucle qui s'exécute **level** fois pour gérer l'indentation en fonction du niveau.
 - **System.out.print("\t")** : À chaque itération, imprime un caractère de tabulation (**\t**) pour indenter la ligne courante, ce qui visuellement représente le niveau de profondeur dans l'arbre.
- Affichage du nom et prénom (**System.out.println(root.getNom() + " " + root.getPrenom())**) : affiche le nom et le prénom de la **Personne** actuelle (**root**) sur la console.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (25)

Description détaillée de la méthode **printHierarchyTree** (2) :

- **List<Personne> enfants = root.getEnfants()** : récupère la liste des enfants de **root** et l'assigne à la variable **enfants**.
- Affichage récursif des enfants :
 - **for(Personne p : enfants)** : itère sur chaque enfant de **root**.
 - **printHierarchyTree(p, level + 1)** : appelle récursivement **printHierarchyTree** sur chaque enfant **p**, augmentant le niveau de profondeur (**level + 1**). Cela affiche chaque enfant avec une indentation supplémentaire.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (26)

Description détaillée de la méthode **printHierarchyTree** (3) :

En résumé, la méthode **printHierarchyTree** affiche l'arbre généalogique de manière récursive en partant d'une **Personne** donnée (la racine). Elle imprime le nom et le prénom de la personne avec une indentation correspondant à son niveau de profondeur dans l'arbre, puis appelle récursivement la méthode pour chacun de ses enfants, augmentant le niveau de profondeur à chaque appel récursif. Cela crée une représentation hiérarchique visuelle de l'arbre sur la console.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (27)

createTree (statique) :

- Niveau d'accès : public
- Type de retour : Arbre
- Description : cette méthode crée un arbre en lisant les données depuis un fichier CSV et en construisant l'arbre de hiérarchie.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (28)

Description détaillée de la méthode **createTree** :

- Création d'une instance d'**Arbre** (**Arbre arbre = new Arbre()**) : crée une nouvelle instance de la classe **Arbre** et l'assigne à la variable **arbre**.
- Lecture des données depuis un fichier CSV (**arbre.readData("\\src\\familles.csv")**) : appelle la méthode **readData** sur l'instance **arbre** en passant par le chemin du fichier CSV **"\\src\\familles.csv"**. Cette méthode lit les données du fichier CSV et initialise la map **personnesMap** et la liste **roots** avec les données lues.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (29)

Description détaillée de la méthode **createTree** (2) :

- Construction de l'arbre hiérarchique :
 - **for(Personne p : arbre.getRoots())** : itère sur chaque **Personne** dans la liste **roots** de l'instance **arbre**. **getRoots** retourne la liste des racines (personnes sans père ou mère).
 - **arbre.buildHierarchyTree(p)** : pour chaque racine **p**, appelle la méthode **buildHierarchyTree** pour construire de manière récursive l'arbre de hiérarchie à partir de cette racine.



III. Visualisation de l'arbre (Interface Java)


Programmation orientée objet

Classe Arbre : Méthodes (30)

Description détaillée de la méthode **createTree** (3) :

- Retour de l'arbre construit (**return Arbre**) : retourne l'instance **arbre** maintenant remplie et structurée avec les données lues et l'arbre hiérarchique construit.

En résumé, la méthode **createTree** crée une nouvelle instance de la classe **Arbre**, lit les données depuis un fichier CSV spécifié, construit l'arbre hiérarchique en appelant récursivement **buildHierarchyTree** sur chaque racine identifiée et retourne l'instance de l'arbre ainsi construite. Cela permet de créer et d'initialiser un arbre généalogique complet à partir des données d'un fichier CSV.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (31)

ajouterIndividu :

- Niveau d'accès : public
- Type de retour : void
- Paramètres :
 - Variable id de type String
 - Variable nom de type String
 - Variable prenom de type String
 - Variable dateNaissance de type String
 - Variable nationalite de type String
 - Variable idPere de type String
 - Variable idMere de type String
 - Variable pathname de type String
- Description : cette méthode ajoute un individu dans le fichier CSV spécifié.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (32)

Description détaillée de la méthode **ajouterIndividu** :

- Bloc **try** : commence un bloc **try** pour capturer et gérer les exceptions qui pourraient survenir lors de l'écriture dans le fichier.
- Création d'un objet **File** (**File file = new File(pathname)**) : crée un objet **File** avec le chemin spécifié (**pathname**).



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (33)

Description détaillée de la méthode **ajouterIndividu** (2) :

- Ouverture du fichier CSV en mode ajout :
 - **new FileWriter(file, true)** : crée un **FileWriter** pour le fichier, en mode ajout (append mode) grâce au deuxième argument **true**. Cela signifie que les nouvelles données seront ajoutées à la fin du fichier sans écraser les données existantes.
 - **new CSVWriter(new FileWriter(file, true), ',', CSVWriter.NO_QUOTE_CHARACTER, CSVWriter.DEFAULT_ESCAPE_CHARACTER, CSVWriter.DEFAULT_LINE_END)** : crée un **CSVWriter** en utilisant le **FileWriter**. Les paramètres spécifient que le séparateur de champs est une virgule, qu'il n'y a pas de caractères de citation autour des champs, qu'il utilise le caractère d'échappement par défaut et la fin de la ligne par défaut.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (34)

Description détaillée de la méthode **ajouterIndividu** (3) :

- Création du tableau de données (**String[] data = {id, nom, prenom, dateNaissance, nationalite, idPere, idMere}**) : crée un tableau de chaînes de caractères contenant les informations de la personne à ajouter.
- Écriture des données dans le fichier CSV (**writer.writeNext(data)**) : écrit une nouvelle ligne dans le fichier CSV avec les données fournies dans le tableau **data**.
- Fermeture du **CSVWriter** (**writer.close()**) : ferme le **CSVWriter** pour libérer les ressource associées au fichier.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (35)

Description détaillée de la méthode **ajouterIndividu** (4) :

- Bloc **catch** :
 - **catch(IOException e)** : capture les exceptions du type **IOException** qui peuvent survenir pendant l'écriture du fichier.
 - **e.printStackTrace()** : imprime la pile d'appels de l'exception pour aider à diagnostiquer les erreurs.




III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (36)

Description détaillée de la méthode **ajouterIndividu** (5) :

En résumé, la méthode **ajouterIndividu** ajoute une nouvelle personne dans le fichier CSV spécifié sans effacer les données existantes. Elle crée un **FileWriter** en mode ajout et un **CSVWriter** pour écrire les données. Elle assemble les informations de la personne dans un tableau de chaînes de caractères, écrit ce tableau comme une nouvelle ligne dans le fichier CSV et ferme ensuite le **CSVWriter** pour s'assurer que les ressources sont correctement libérées. Si une erreur se produit pendant ce processus, elle est capturée et imprimée.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (37)

supprimerIndividu :

- Niveau d'accès : public
- Type de retour : void
- Paramètres :
 - Variable id de type String
 - Variable pathname de type String
- Description : cette méthode supprime un individu dans le fichier CSV spécifié.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (38)

Description détaillée de la méthode **supprimerIndividu** :

- Bloc **try** : commence un bloc **try** pour capturer et gérer les exceptions qui pourraient survenir lors de l'écriture dans le fichier.
- Lecture du fichier CSV (**CSVReader reader = new CSVReader(new FileReader(pathname))**) : crée un **CSVReader** pour lire les données du fichier CSV spécifié par **pathname**.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (39)

Description détaillée de la méthode **supprimerIndividu** (2) :

- Lecture de toutes les lignes :
 - **List<String[]> allLines = reader.readAll()** : lit toutes les lignes du fichier CSV et les stocke dans une liste de tableaux de chaînes de caractères (**List<String[]>**).
 - **reader.close()** : ferme le **CSVReader** pour libérer les ressources associées au fichier.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (40)

Description détaillée de la méthode **supprimerIndividu** (3) :

- Filtrage des lignes :
 - **List<String[]> filteredLines = allLines.stream()** : crée un flux (stream) à partir de la liste **allLines**.
 - **.filter(line -> !line[0].equals(id))** : applique un filtre pour éliminer les lignes dont le premier élément (l'identifiant) est égal à **id**. Cela signifie que seules les lignes qui ne correspondent pas à l'identifiant à supprimer sont conservées.
 - **.collect(Collectors.toList())** : collecte les résultats filtrés dans une nouvelle liste (**filteredLines**).



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (4)

Description détaillée de la méthode **supprimerIndividu** (4) :

- Réécriture du fichier CSV (**CSVWriter writer = new CSVWriter(new FileWriter(pathname))**) : crée un **CSVWriter** pour écrire dans le fichier CSV spécifié par **pathname**. Cette fois, le fichier est ouvert en mode écriture (et non en mode ajout), ce qui écrase le contenu existant.
- Écriture des lignes filtrées :
 - **writer.writeAll(filteredLines)** : écrit toutes les lignes de la liste **filteredLines** dans le fichier CSV.
 - **writer.close()** : ferme le **CSVWriter** pour libérer les ressources associées au fichier.



III. Visualisation de l'arbre (Interface Java)

Programmation orientée objet

Classe Arbre : Méthodes (42)

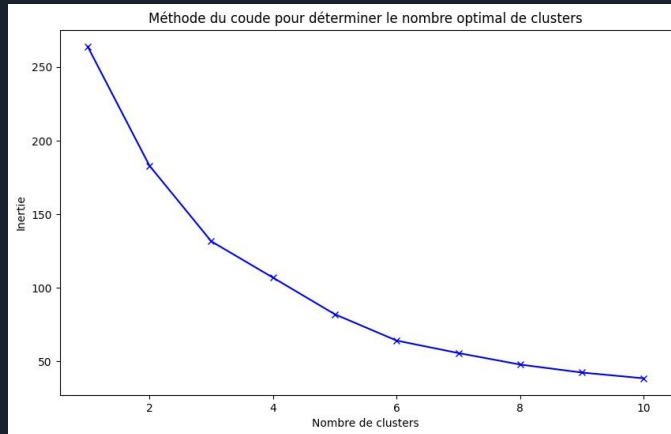
Description détaillée de la méthode **supprimerIndividu** (5) :

- Bloc **catch** :
 - **catch(IOException e)** : capture les exceptions de type **IOException** qui peuvent survenir pendant la lecture ou l'écriture du fichier.
 - **e.printStackTrace()** : imprime la pile d'appels de l'exception pour aider à diagnostiquer les erreurs.

En résumé, la méthode **supprimerIndividu** supprime une personne du fichier CSV en lisant toutes les lignes, en filtrant celles qui ne correspondent pas à l'identifiant fourni, puis en réécrivant le fichier avec les lignes filtrées. Cela garantit que l'individu spécifié par l'identifiant est supprimé du fichier. Si une erreur se produit pendant ce processus, elle est capturée et imprimée.

IV. Le diabète et programme python

- base de donnée précédente avec des valeurs supplémentaires
- données simplifiés
- Indice de masse corporelle, insuline, glucose,outcome



1	ID,Nom,Prénom,Date_de_naissance,Glucose,Insulin,IMC,Outcome
2	1,"Dubois","Alice","1970-05-15",148, 0, 33.6, 1
3	2,"Dubois","Louis","1971-07-20",85, 0, 26.6, 0
4	3,"Dubois","Charlotte","1993-09-10",183, 0, 23.3, 1
5	4,"Dubois","Hugo","1995-11-25",89, 94, 28.1, 0
6	5,"Dubois","Emma","1997-02-18",137, 168, 43.1, 1
7	6,"Dubois","Gabriel","1999-04-30",116, 0, 25.6, 0
8	7,"Dubois","Chloe","2000-06-05",78, 88, 31, 1
9	8,"Dubois","Lucas","2022-08-12",115, 0, 35.3, 0
10	9,"Dubois","Eva","2024-10-28",167, 543, 30.5, 1



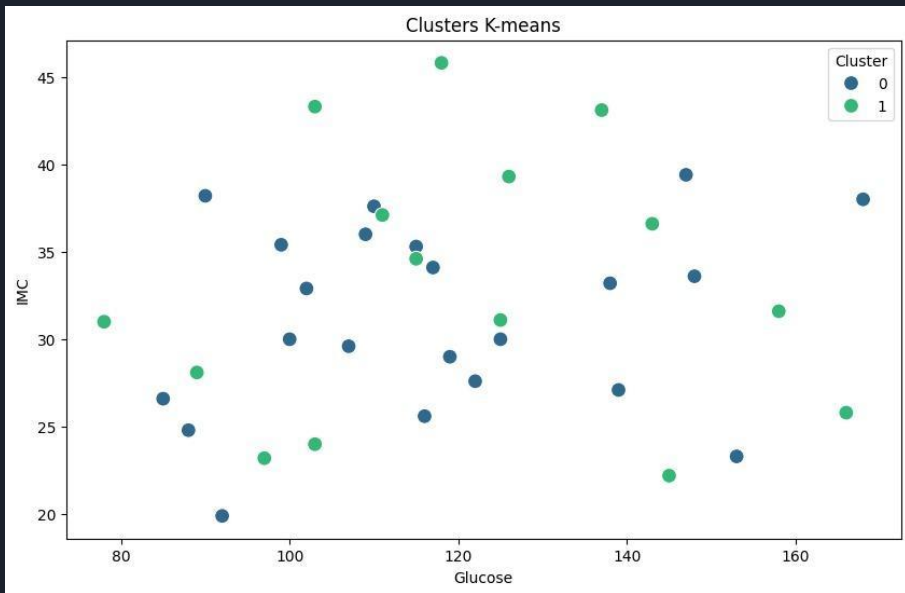
IV. Le diabète et programme python

- algorithme K-means, choix de cluster

- **La distance Euclidienne** : C'est la distance géométrique qu'on apprend au collège. Soit une matrice X à n variables quantitatives. Dans l'espace vectoriel E^n . La distance euclidienne d entre deux observations x_1 et x_2 se calcule comme suit :

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^n (x_{1j} - x_{2j})^2}$$

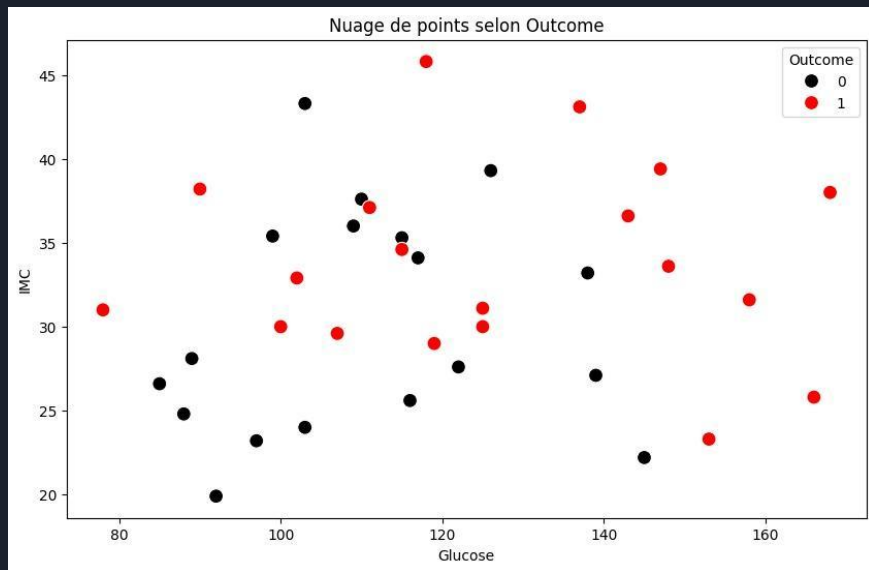
IV. Le diabète et programme python



Ce graphique montre une visualisation des clusters obtenus avec l'algorithme du K-means. Les points de données sont projetés sur deux composantes principales (IMC et Glucose), ce qui permet de visualiser les clusters dans un espace bidimensionnel.

Ces clusters représentent des groupes de membres partageant des similitudes dans leurs antécédents de diabète et d'autres facteurs de risque. Ainsi, chaque cluster permet d'identifier les membres de la famille ayant des caractéristiques similaires et d'évaluer leur risque de développer un diabète.

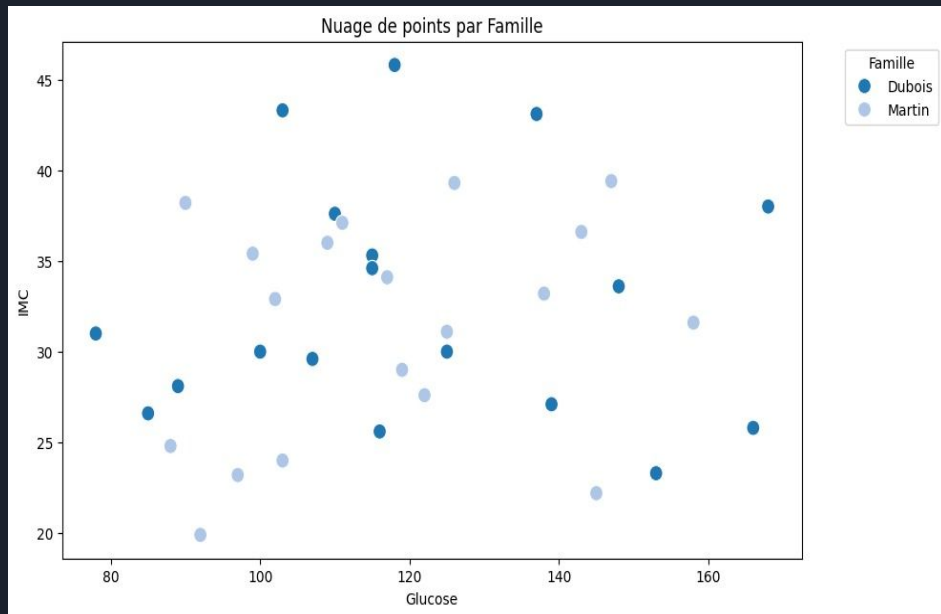
IV. Le diabète et programme python



Ce graphique est le même que le précédent à la différence que celui-ci montre uniquement qui a le diabète.

0 : la personne n'a pas le diabète,
1 : la personne a le diabète

IV. Le diabète et programme python



Cluster 1 : représente un groupe à risque élevé de diabète.

Taux de glucose et d'insuline soient bien gérés et l'IMC plus proche de la moyenne.

Cluster 0 : représente un groupe à risque moins élevé de diabète.

IMC proche de la moyenne et un taux de glucose et d'insuline dans la norme.



V. Conclusion

- > Création d'arbre généalogique = beaucoup de défis

 - > collecte informations

 - > Définir objectif

- > Arbre généalogique pas simple:

 - > Vérification liens entre générations

 - > création de la base de données

- > Arbre généalogique dans le domaine de la médecine:

 - > Facilité à détecter les maladies

 - > Permet de mieux comprendre les maladies génétique

 - > Emettre des hypothèses sur des potentiels maladies héréditaires