POLYTECH SORBONNE

MAIN4

Base de Données

TD 1 : Conception

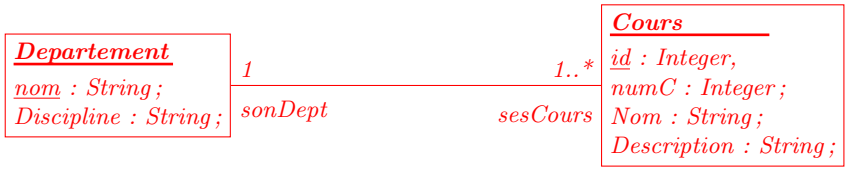
(Correction)

Exercice 1

Donnez un diagramme UML pour le sujet suivant en indiquant très précisément les clés.

On considère les ensembles de classes Cours et Departement : Un cours est donné dans un seul département il a un numéro dans son departement, un nom et une description (champs texte). Notez bien que différents départements peuvent offrir des cours de même numéro. Chaque département a un nom unique et une discipline (champs texte). Par exemple, le cours 1 du département Informatique a pour nom *BD* alors que le cours 1 du département Biologie a pour nom *Biologie végétale*.

Correction :

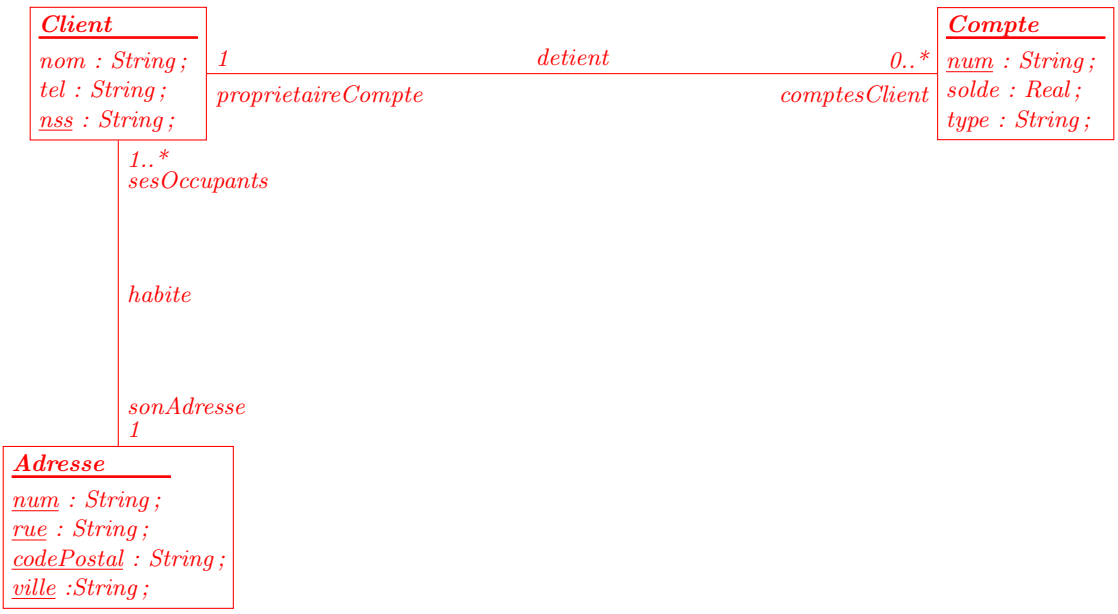


Exercice 2

Nous souhaitons concevoir une base de données pour une banque incluant des informations sur des comptes et des clients. L'information relative à un client consiste en son nom, son adresse, son téléphone ainsi que son numéro de sécurité sociale. Les comptes ont un numéro, un solde, un type (courant, épargne etc...). Il faut en outre pouvoir déterminer à tout moment quels sont les comptes d'un client donné ainsi que le client propriétaire d'un compte.

1. Donner un diagramme de classes UML.
2. Discutez les cardinalités minimales possibles des associations.

Correction :



Un numéro de sécurité sociale peut contenir des lettres (les gens nés en Corse (2A, 2B) essentiellement), donc String plutot que Integer. De la même manière, un téléphone peut contenir des symboles (par exemple + pour rentrer un indicatif de pays). De plus, un numéro de téléphone peut commencer par un 0 significatif (ce que ne permet pas de représenter Integer).

Pour les comptes, on peut imaginer que le numéro du compte est celui qui est sur le RIB, il peut donc y avoir des lettres ou des 0 significatifs (on préfère donc utiliser String plutot que Integer).

Le solde peut avoir des décimales, on utilise plutot Real. Pour le type, on ne se prend pas la tête et on utilise String. Integer aurait pu être accepté.

Comme vu en cours, différentes possibilités existent : l'adresse en tant qu'attributs du client, l'adresse séparée (ci-dessus), ou même des classes PAYS, VILLE, CODE_POSTAL, ADRESSE si on veut maintenir une base de noms de pays, villes, etc., par exemple pour faire de la complétion automatique ou du remplissage semi-automatique de formulaires.

Exercice 3

Modifier la conception de l'exercice précédant en tenant compte des informations supplémentaires suivantes :

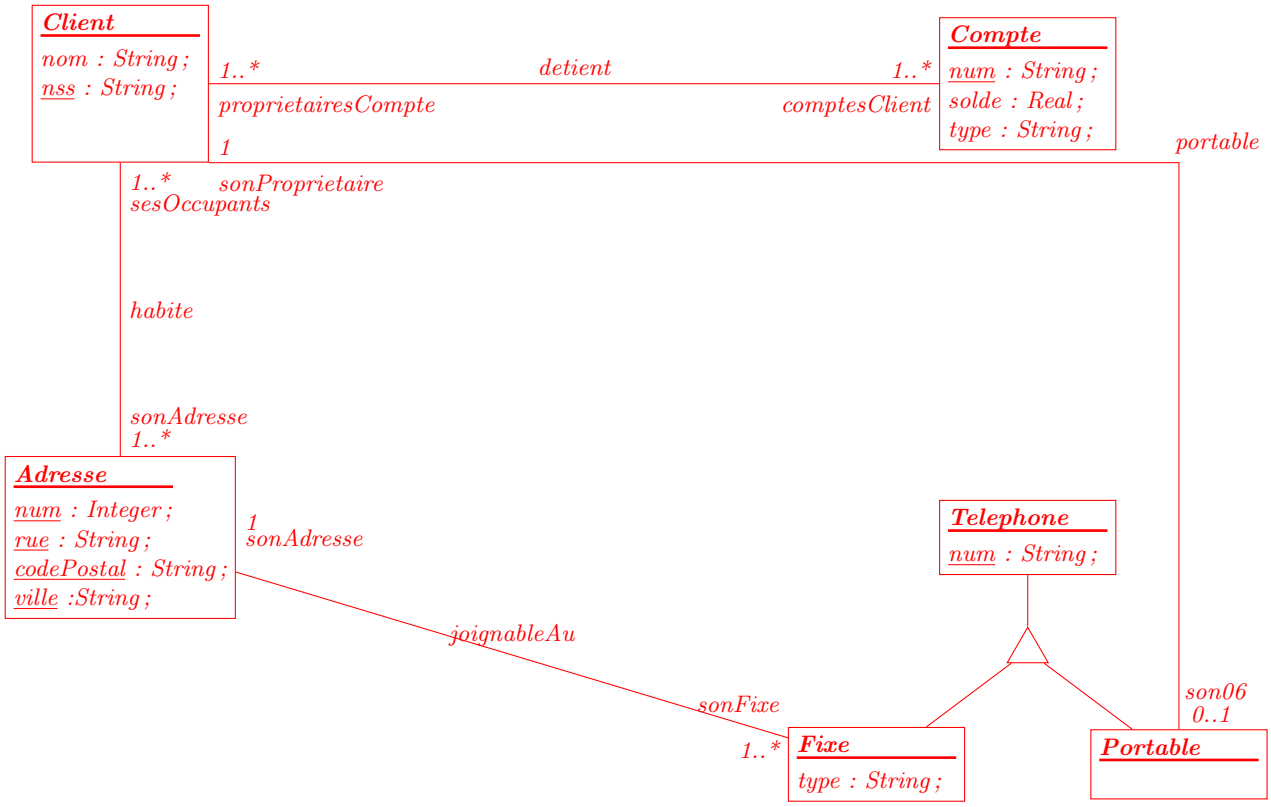
1. Pour chaque compte, il peut y avoir plusieurs titulaires.
2. Un client peut posséder plusieurs comptes mais il est considéré comme client s'il a au moins un compte.
3. Les adresses sont composées d'un numéro, d'un nom de rue, d'une ville et d'un code postal. Un client peut avoir plusieurs adresses.
4. Pour chaque adresse, il peut y avoir plusieurs numéros de téléphone fixe et on connait leur fournisseur (*Free*, *Orange*, etc). Ceci implique qu'il est nécessaire de conserver pour chaque adresse quels numéros lui sont associés. En outre, chaque client peut avoir un numero de telephone portable (on n'en stockera qu'un seul au plus) dont il est le seul propriétaire.

Correction :

1. *On change la cardinalité (et le nom) du rôle* `proprietaire(s)Compte`
2. *On change la cardinalité du rôle* `comptesClient` *pour devenir au moins 1*

3. *On supprime l'attribut adresse et on rajoute une association* `habite` *qui relie la classe Client et la classe Adresse, avec les bonnes cardinalités (on suppose que plusieurs clients peuvent habiter à la même adresse, par exemple des conjoints qui ont des comptes séparés). Dans les deux cas il semble raisonnable qu'un client ait au moins une adresse et qu'une adresse soit au moins associée à un client. (On utilise String pour codePostal à cause du 0 significatif. On pourrait aussi l'utiliser pour num pour prendre en compte les bis, ter, ...). De plus comme rien est unique a priori dans l'adresse, on met tout en PK.*
4. *On utilise de l'héritage pour différencier les téléphones portables et les téléphones fixes. On remarque que la classe* `Telephone` *n'est pas directement utilisée mais est juste la pour regrouper les attributs communs aux deux types de téléphones. On aurait aussi pu dériver les téléphones fixes en plusieurs sous-catégories, mais on choisit la solution alternative de mettre le type sous forme de chaine de caractères. C'est l'approche classique quand tous les types sont les mêmes et que l'on ne sait pas a priori à l'avance combien on va en avoir.*

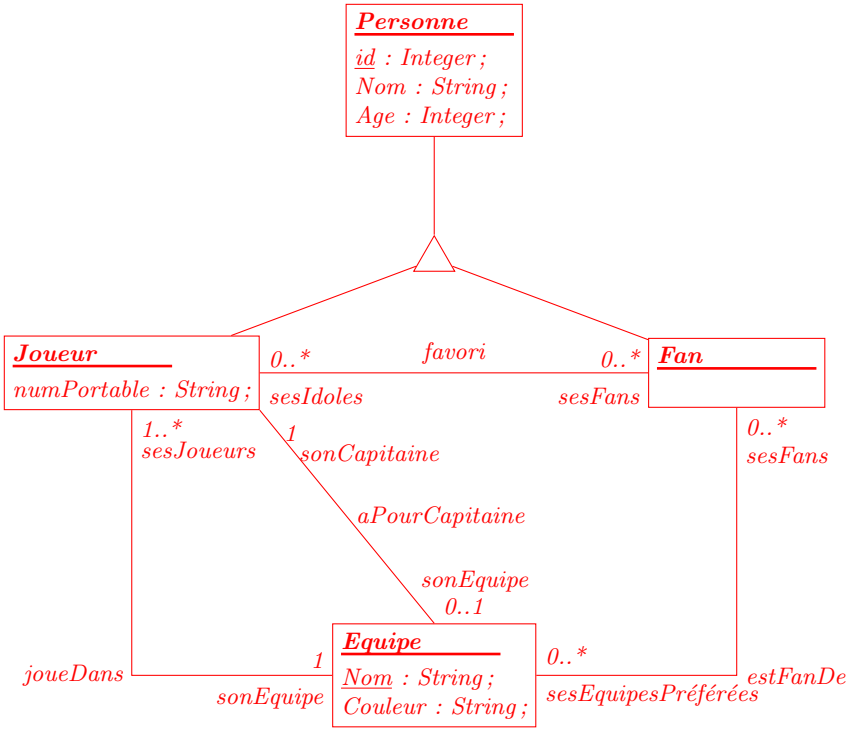
Le diagramme de classe complet :



Exercice 4

Donner une représentation UML pour la gestion d'une base de données stockant de l'information relative à des équipes, des joueurs et leurs fans. On suppose que :

- chaque équipe a un nom, un capitaine et une couleur de maillot.
- chaque joueur a un nom, un âge et un numéro de portable. Un joueur n'appartient qu'à une seule équipe.
- chaque fan a un nom, un âge, des équipes favorites et des joueurs favoris.



Correction :

L'héritage entre `Personne` *et* `Joueur/Fan` *n'est pas du tout indispensable, on aurait pu séparer les deux.*

Exercice 5*

Supposons que l'on souhaite ajouter au schéma de l'exercice 4 les informations sur les matchs disputés par les équipes. Un match a lieu à une certaine date entre deux équipes, et pour chaque équipe, un joueur est capitaine lors de ce match. Nous aurons donc au moins les données suivantes :

(equipe1, equipe2, capitaineEquipe1, capitaineEquipe2, date)

1. Dessinez le diagramme modifié
2. Remplacez l'association que vous venez de créer par un nouvel ensemble de classes et des associations binaires.

Correction : *Optionnel, envoyez-moi votre travail si vous le souhaitez.*
