TP JAVA n°10: Persistance des Objets avec JDBC

Pour effectuer cette séance de TP :

- vous pouvez utiliser la base MySQL dont le script est proposé (32 bits & 64 bits)

- vous pouvez utiliser la base Access proposée sur la zone libre si vous êtes en 32 bits

Manipulation préalable pour l'utilisation d'une base de données MySQL ...

1.Installation de WampServer (si cela n'est pas déjà fait !)

Allez sur le site : http://www.wampserver.com/ , dans la partie **Download** et téléchargez le **WampServer** correspondant à votre environnement (32 bits ou 64 bits).

Installez-le sur votre machine.

2. Base de données MySQL :
Cliquez sur le lien **phpMyAdmin** de **WampServer.**

→2.1 Création de la base **cabinetmedical** :
Cliquez sur l'onglet **Base de données.**
Rentrez le nom que vous souhaitez donner à la base. Pour ce TP, ce sera **cabinetmedical,**
puis cliquez sur le bouton **Créer**.
Si tout se passe bien, votre base est créée et apparaît à la fois dans le menu à gauche et dans le tableau dans la colonne *Base de Données*.

→2.2 Création des tables de la base **cabinetmedical** :
Cliquez sur **cabinetmedical** afin de travailler sur cette base.
Vous pouvez alors créer vous-même les différentes tables de la base, ce qui n'est pas le but de ce TP... Ainsi pour gagner du temps lors de cette séance de TP, un script **cabinetmedical.sql** est disponible sur la zone libre.
Cliquez alors sur l'onglet **Importer.**
Dans la partie ***Fichier à importer*** , sélectionnez le fichier **cabinetmedical.sql** grâce au bouton **Parcourir**.
Cliquez ensuite sur le bouton **Exécuter**.
Les tables **personne** et **adresse** sont alors importées !

3. Driver MySQL :
Pour trouver un driver pour **MySQL**, vous pouvez utiliser le site d'Oracle ...ou vous rendre directement sur le site http://www.mysql.com et récupérer la dernière version du driver **MySQL Connector/J** qui est le driver officiel **JDBC** pour **MySQL**. Ce driver peut être téléchargé à l'adresse suivante:
http://www.mysql.com/downloads/connector/j/

4. Mise en place du driver dans le projet Java :
Une documentation sur le driver **Connector/J** est proposé sur le site mysql.com dans l'onglet **Documentation** dans la partie **20.3 MySQL Connector/J** qui est accessible via l'adresse http://dev.mysql.com/doc/refman/5.5/en/connector-j.html
La partie 21.3.2. Connector/J Installation concerne la mise en place du driver dans le projet Java.

→4.1 Si vous venez de charger le driver depuis le site mysql.com, dézippez l'archive et récupérer le fichier **jar** : `mysql-connector-java-[version]-bin.jar`.

→4.2 Vous devez ensuite ajouter ce **jar** à votre **classpath**. La documentation indique également que pour utiliser ce driver, il est nécessaire d'utiliser la classe `com.mysql.jdbc.Driver` comme classe qui implémente `java.sql.Driver` c-a-d que c'est cette classe qui devra être passée en paramètre du **Class.forName**

5. Obtenir une connexion avec la base cabinetMedical
La partie 21.3.3. Connector/J Examples de la documentation sur le driver **Connector/J** vous propose des exemples qui permettent d'établir une connexion avec une base et notamment l'exemple Example 21.1, "Connector/J: Obtaining a connection from the `DriverManager`" (http://dev.mysql.com/doc/refman/5.5/en/connector-j-usagenotes-basic.html#connector-j-examples-connection-drivermanager)

A partir de cette documentation, on détermine l'**url** d'accès à notre base **cabinetmedical** qui doit être de la forme : `jdbc:mysql://localhost/cabinetmedical`
Si vous venez d'installer **WampServer** et de lancer votre **PhpMyAdmin**, votre login sera : `root`
et vous n'avez pas de mot de passe à spécifier.
Pour déterminer votre login dans **PhpMyAdmin**, cliquez dans le bandeau du haut sur **localhost.**
Dans le cadre **SQL**, consulter alors le paramètre **Utilisateur.** Si sa valeur est : `root@localhost`
alors votre login sera **root** !

Ainsi les paramètres de la méthode **getConnection** pour une base **cabinetMedical** sous MySQL seront:

```
String url = "jdbc:mysql://localhost/cabinetmedical";
String login = "root";
String password = "";
```

Manipulation préalable pour l'utilisation d'une base de* données `Access` *(32 bits)

Dans ce TP, nous allons accéder localement à une Base de Donnée `Access` en utilisant un driver de type pont JDBC/ODBC. Avant tout, il est nécessaire de passer par l'étape suivante :

Enregistrer une base de données dans ODBC

1. Récupération de la base de données

➢ Dans un premier temps, vous devez créer ou récupérer la base de données sur laquelle vous souhaitez travailler.

Créer un répertoire **BD** dans votre répertoire **JAVA** et copier la base **`cabinetMedical.mdb`** qui se trouve sur la zone libre.

*Pour pouvoir utiliser JDBC, il faut un pilote qui est spécifique à la base de données à laquelle on veut accéder. Avec le JDK, Sun fournit un pilote qui permet l'accès aux bases de données via **ODBC**. Pour utiliser un pilote de type 1 (pont ODBC−JDBC), il est nécessaire d'enregistrer la base de données dans ODBC avant de pouvoir l'utiliser.*

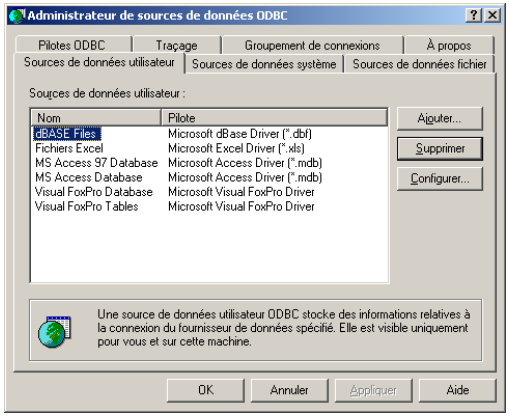
2. Mise en place du lien ODBC/Enregistrer une base de données dans ODBC

Pour enregistrer une nouvelle base de données, il faut utiliser **l'Administrateur de Source de Données ODBC**.

➢ Pour lancer cette application, allez dans le **panneau de configuration** et cliquez sur **Outils d'administration**

Puis cliquer sur **Sources de données ODBC**

➢ La fenêtre **Administrateur de Sources de données ODBC** s'ouvre.

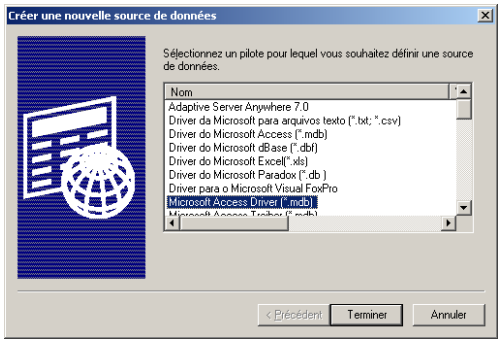


L'outil se compose de plusieurs onglets.

↳ L'onglet "**Pilote ODBC**" liste l'ensemble des pilotes qui sont installés sur la machine.

↳ L'onglet "**Source de données utilisateur**" liste l'ensemble des sources de données pour l'utilisateur couramment connecté sous Windows.

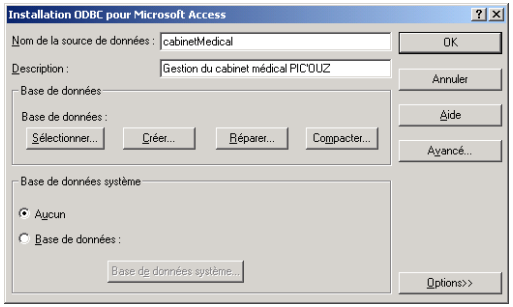
➢ Le plus simple est de créer une telle source de données en cliquant sur le bouton "**Ajouter**".
➢ Une boite de dialogue **Créer une Nouvelle Source de Données** s'ouvre.
Elle permet de sélectionner le pilote qui sera utilisé par la source de données.



↳ pour notre application, nous sélectionnerons **Microsoft Access Driver (*.mdb)**

↳ cliquer sur **Terminer.**

➢ Comme le pilote sélectionné concerne une base Microsoft Access, la boîte de dialogue **Créer une Installation ODBC pour Microsoft Access** s'ouvre.



Il suffit de saisir les informations nécessaires notamment le nom de la source de données et de sélectionner la base.

↳ Donner le **nom à la source de données**, par exemple : **`cabinetMedical`**
Ce nom sera utilisé dans le code Java comme élément de l'url de connexion.

↳ Compléter la **description (facultatif)**

↳ Cliquez sur **Sélectionner** et indiquer le chemin vers le fichier physique en sélectionnant la base de données **`cabinetMedical.mdb`** dans votre répertoire **Z:\JAVA\BD**

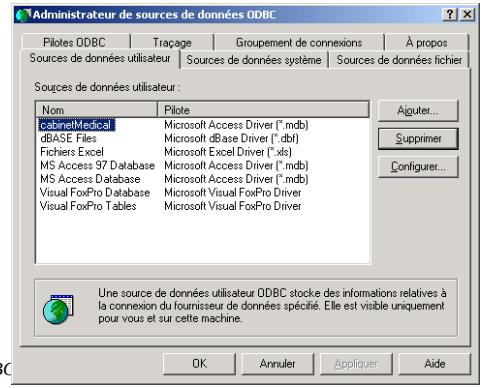
↳Cliquez sur le bouton **Ok** pour créer la source de données qui pourra alors être utilisée.

La base **`cabinetMedical`**

apparaît désormais sous l'onglet **Sources de données utilisateur.**

Cliquez sur **OK** pour terminer.

Attention, à l'IUT, lorsque vous fermez votre session, vous perdez le lien ODBC… Vous devrez effectuer l'enregistrement de la base dans ODBC à chacune de vos connections !!!



Exercice 1 : *Classe* `SimpleConnection`

Ecrire la classe **`SimpleConnection`** vue en TD dans le paquetage **`com.iut.cabinet.util`**

Afin de tester le bon fonctionnement la classe **`SimpleConnection`**, rajouter la méthode suivante :

```
public static void main(String[] args) {
          Connection co1=null;
          try {
               co1 = SimpleConnection.getInstance().getConnection();
          } catch (CabinetTechniqueException e) {
               e.printStackTrace();
          }
          System.out.println("-->Simple Connection: "+co1.toString());
     }
```

Exécuter la classe **`SimpleConnection`**.

…. Si tout est OK, vous pouvez continuer ! ! !

… Pour ce qui concerne la journalisation,
*on vous demande de ne réaliser vos **logs qu'en fin de TP** (exercice 5).*
Dans un premier temps, pour ne pas alourdir le code,
continuez donc l'énoncé de TP sans écrire les logs,
vous les rajouterez par la suite …

Exercice 2 : Modification de l'existant pour pouvoir utiliser la base **`cabinetMedical.mdb`** ***dans le*** **`projetCabinetMedical`**…

Pour les explications concernant le mapping des classes `Adresse` et `Personne`, reportez-vous aux explications fournies dans l'énoncé de l'exercice n°2 dans l'énoncé du TD.

↳ 2.1. **Modification de la classe : `Adresse.java`**

Adresse

idAdresse	numero	rue	voie	batiment	codePostal	ville	pays	idPersonne
1	15	avenue Jean Jaure			87000	Limoges	France	1
18	3	rue de Limoges			87170	Isle	France	2
19	10	rue de Toulouse		Batiment A	87000	Limoges	France	3
20	123Bis	Boulevard d'Ici			87000	Limoges	France	4

➢ ***Travail à faire :*** Pour pouvoir sérialiser les données dans la base Access **`cabinetMedical.mdb`**, et plus précisément dans la table `Adresse`, vous devez rajouter dans la classe `Adresse` du paquetage `java.iut.com.metier`, un nouvel attribut ***privé*** de type **`Integer`** que vous appellerez **`idAdresse`** et qui correspondra bien sûr à l'identifiant technique de la table.
Vous générerez également ses ***getteur*** *et* ***setteur*** (sous Eclipse, génération automatique à partir du menu `Source → Generate Getters and Setters`)

➢ Remarque : il n'est pas nécessaire pour l'instant de modifier les constructeurs de la classe `Adresse`, (vous pourrez le faire chez vous pour avoir un programme « plus propre ») .
Lorsque vous instancierez un nouvel objet de type `Adresse` l'attribut `idAdresse` prendra la valeur par défaut c-a-d `null`. S'il est nécessaire de mettre à jour `idAdresse`, il suffira alors d'appeler le setteur…

➢ La mémorisation du champ `idPersonne` dans la table `Adresse` sous Access n'entraîne bien sûr aucune modification dans la classe `Adresse` sous java.

↳ 2.2. **Pas de modification des classes : `Personne.java, Patient.java, Professionnel.java`**

Personne

idPersonne	nom	prenom	datenaissan	male	telephone	portable	email	idAscendan	idTypePersc	nir	medecinTra	immatricula	specialite
1	DUPONT	Julie	21/05/1960	☐	0555434355	0606060606	julie.dupont@		0	2600587001123	LEDOC Paul		
2	LEDOC	Paul	10/07/1976	☑	0555434343	0612345678	paul.ledoc@le		1			871255358	généraliste
3	CHILDREN	Rose	16/02/1970	☐	0555434343	0678654321	rose.children@		1			312444555	pédiatrie
4	DURAND	Alfred	23/05/1968	☑	0512348989	0605050505	alfred.durant@		0	168072B12345(	LEDOC Paul		

➢ Il est à noter qu'aucune modification ne doit être apportée dans le projet `cabinetMedical` sur les classes `Personne`, `Patient` et `Professionnel` pour que ce mapping soit correct.

➢ Il est juste nécessaire de bien avoir à l'esprit la règle d'identification des instances de type `Patient` ou `Professionnel` liée à la valeur du champ `idTypePersonne` de la table Access
- lorsque le champ `idTypePersonne` a pour valeur **`0`**, l'objet correspondant (à cet enregistrement) dans le programme Java est une instance de la classe **`Patient`**
- lorsque le champ `idTypePersonne` a pour valeur **`1`**, l'objet correspondant (à cet enregistrement) dans le programme Java est une instance de la classe **`Professionnel`**

Exercice 3: *Mise en place d'une première requête sur la base :*
méthode *findAllPersonne* de la classe *PersonneDAOJDBC*

3.1. Mise en place des DAO de l'application.

➢ Sur la zone libre, vous pouvez récupérer la classe AdresseDAOJDBC et l'importer dans le package com.iut.cabinet.metier. Dans cette classe, la méthode suivante a été implémentée :

```
public static Adresse findAdresseByIdPersonne(Integer idPersonne, Connection c)
                    throws CabinetTechniqueException
```

Elle vous sera nécessaire pour implémenter la requête **findAllPersonne** de la classe PersonneDAOJDBC
… Libre à vous de compléter ce DAO plus tard …

➢ Créer dans le paquetage com.iut.cabinet.metier, la classe PersonneDAOJDBC et implémenter la méthode écrite en TD:

```
public static Collection<Personne> findAllPersonne(Connection c)
                    throws CabinetTechniqueException
```

Si cette méthode n'a pas été totalement écrite en TD, reprenez l'énoncé de TD pour suivre la démarche et procédez à son implémentation …

➢ Afin de tester votre code, rajouter à la classe PersonneDAOJDBC, la méthode main suivante :
Ce code est disponible sur la zone libre dans le fichier mainFindAllPersonne.txt

```
public static void main(String args[])
{
      Connection conn=null;
      Collection<Personne> maListe=null;
      try {
            conn = SimpleConnection.getInstance().getConnection();
            maListe = findAllPersonne(conn);
      } catch (CabinetTechniqueException e) {
            e.printStackTrace();
      }

      System.out.println("------------------------------");
      System.out.println(" La collection chargée donne : ");
      System.out.println(maListe);
      System.out.println("------------------------------");

      // Il faut valider la transaction
      // car la connection est mode transactionnel : setAutoCommit(false);
      try {
            conn.commit();
      } catch (SQLException e) {
            e.printStackTrace();
      }

      // fermeture de la ressource ouverte c-a-d la connection
      try {
            if (conn!=null && !conn.isClosed()) conn.close();
      } catch (SQLException e) {
            e.printStackTrace();
      }
}
```


3.2. Appel du DAO *PersonneDAOJDBC* dans la méthode listerPatients du contrôleur

Rappelons notre choix de conception pour l'application CabinetMedical :

a. chaque méthode du **Contrôleur** devra ouvrir une nouvelle connexion.
b. La connexion sera ensuite passée en paramètre à chacune de(s) méthode(s) du **DAO** appelée(s) au sein de cette méthode.
c. La transaction devra être *validée* (commit) en fin de méthode (une fois toutes les requêtes réalisées) c-a-d que c'est le **Contrôleur** qui "commit"…
…ou s'il y a eu un problème durant cette transaction, le **Contrôleur** doit *annuler* cette transaction par un appel à rollback (toutes les modifications effectuées dans la base durant la transaction en cours seront alors annulées)
d. La connexion sera ensuite fermée en fin de méthode du **Contrôleur**, afin de respecter le principe suivant : *chaque fermeture d'une connexion est à la charge de « celui » qui l'a demandée (ouverte)…*

Modifier de la manière suivante la méthode listerPatients du contrôleur (classe GererPatientCtrl du paquetage com.iut.cabinet.application):

```
public Collection<PatientDTO>listerPatients() throws CabinetTechniqueException,
                                                                    HelperException
    {
        Collection<Personne> maListe=null;

    ////////////////////////////////////////////////////////
    // 1. Récupération de la liste de Personne provenant de la base …
    ////////////////////////////////////////////////////////
        Connection conn=null;

        // a. Ouverture de la connexion dans le cas d'une BD
        //    dans le cas d'un fichier connexion serait à null...
        conn = SimpleConnection.getInstance().getConnection();

        // b. Appel de la méthode du DAO
        // en passant la connexion en paramètre
        maListe = PersonneDAOJDBC.findAllPersonne(conn);


        //c. La transaction doit être validée
        try {
            conn.commit();
        } catch (SQLException e) {
            throw new CabinetTechniqueException ("Pb lors de la validation
                                                 de la transaction"+e.getMessage());
        }



        //d. La connexion est ensuite fermée
        try {
            if (conn!=null && !conn.isClosed()) conn.close();
        } catch (SQLException e) {
            throw new CabinetTechniqueException ("Pb lors de la fermeture
                                                  de la connection"+e.getMessage());

        }


    ////////////////////////////////////////////////////////
    // 2. Création de la liste de PatientDTO
    ////////////////////////////////////////////////////////
    // ce code ne change pas par rapport à ce que vous aviez écrit avec les fichiers

        Collection<PatientDTO> maListeDTO = new ArrayList<PatientDTO>();

        for(Personne unePersonne : maListe)
        {
            if (unePersonne instanceof Patient)
            {
            PatientDTO unDTO;
            unDTO = HelperPatient.toPatientDTO((Patient)unePersonne);
            maListeDTO.add(unDTO);
            }
        }

        return maListeDTO;
    } // fin listerPatients
```

code à rajouter

Remarque : Il est évident que si vous décidez de travailler uniquement sur des bases de données relationnelle, vous pourrez optimiser ce code plus tard, en écrivant directement dans un DAO, une méthode `findAllPatients` (puisqu'avec une BDR, il est possible d'extraire uniquement des enregistrements de type `Patient` ou de type `Professionnel` en consultant le champ `idTypePersonne`, ce qui n'était pas le cas avec les fichiers où on récupérait une `Collection` de `Personne`(s) dans son ensemble, sans discrimination possible).

3.3. Test de votre application

Pour tester votre code de consultation de la base, vous pouvez dès maintenant relancer indifféremment *sans aucune autre intervention dans votre code* (intérêt de la programmation par couches) :

- soit l'application en mode graphique et plus particulièrement l'IHM `PanelListerPatients`
- soit l'application en mode console `GererPatientIHM` (avec l'option Lister tous les Patients)

Dans les deux cas, vous devez désormais afficher les données en provenance de la base `cabinelMedical.mdb`.

Exercice 4: *Ajout d'une* `Personne` *dans la base de données*

Sur le même principe que l'exercice précédent …

4.1. Mise en place du `Create` du `CRUD` dans le DAO `PersonneDAOJDBC`

➢ Implémenter dans la classe `PersonneDAOJDBC` , la méthode suivante qui permet d'insérer dans les tables de la base `CabinetMedical` un nouvel objet de type `Personne` du programme Java (utilisez de préférence des `PreparedStatement`)

```
public static void storePersonne(Personne unePers, Connection c)
                                    throws CabinetTechniqueException
```

Remarque concernant la ***récupération de la clé primaire*** :
Comme l'indique l'annexe IV du cours sur la persistance des objets dans une base relationnelle, l'interface `Statement` contient une méthode `getGeneratedKeys()` qui renvoie toutes les clés générées par l'ordre SQL. Malheureusement, il semblerait que peu de drivers implémentent la méthode `getGeneratedKeys()`, et à priori pas le driver `jdbc/odbc` que nous utilisons … Il faut donc trouver une autre solution pour récupérer l'identifiant technique du dernier enregistrement créé.

4.2. Appel du DAO `PersonneDAOJDBC` dans la méthode `creerPatient` du contrôleur

En vous inspirant de l'exercice précédent, modifier la méthode `creerPatient` du contrôleur (classe `GererPatientIHM`)afin qu'elle appelle désormais la méthode `storePersonne` du DAO `PersonneDAOJDBC`.

4.3. **Test de votre application**

Pour tester votre code, vous pouvez dès maintenant relancer indifféremment *sans aucune autre intervention dans votre code* :

- soit l'application en mode graphique et plus particulièrement l'IHM `PanelCreerPatient`
- soit l'application en mode console `GererPatientIHM` (avec l'option Créer un Patient)

Exercice 5: *Mise en place de la journalisation …*

*… Pensez à rajouter les **logs en entrée et sortie des méthodes** que vous venez d'écrire. Nous vous rappelons également que toutes vos **exceptions techniques** doivent être loguées dans la version finale que vous nous remettrez en fin de module …*

Pour tester la journalisation, vous pouvez, par exemple, introduire une erreur dans votre code en falsifiant avec un "**E**" (au lieu d'un "**e**") le nom du driver par exemple :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriverE");
```

… Vous testerez ainsi le log écrit dans la classe `SimpleConnection` lors d'une `ClassNotFoundException` c-a-d lorqu'une exception technique (`CabinetTechniqueException`) indiquant une erreur de driver est levée.

… une fois ce test effectué, n'oubliez pas de corriger l'erreur qui levait l'exception !!! et de revenir sur un code correcte similaire à

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Améliorations des fonctionnalités de l'application

Quelques pistes …

➢ En vous inspirant de tout ce qui a été fait jusqu'à présent, continuez à programmer les différentes options du use case `GererPatient`. Afin de compléter les opérations du CRUD, il ne vous reste plus qu'à vous intéresser à ces deux parties.

- Implémentation du cas : **Modifier un Patient** du use case `GererPatient`
- Implémentation du cas : **Supprimer un Patient** du use case `GererPatient`

Deux possibilités s'offrent à vous :

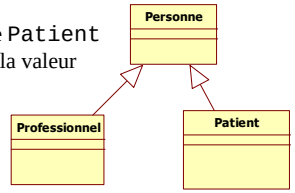
- La plus simple : faire une ***suppression physique*** du `Patient` dans la Base (`delete`)
- La plus "réaliste" : faire une ***suppression logique*** du `Patient`. Dans ce cas-là, vous devrez rajouter un champ dans la base `Personne`… et modifier tout votre code précédent pour tenir compte de la présence ou non du `Patient` dans la base…Dans une application "réelle", la suppression logique serait préférable…

➢ Vous pouvez également vous intéresser à la Prise en compte d'un ascendant (en créant un descendant)
Veuillez consulter **l'annexe I** qui vous explique comment doit être mis en place le mapping objet/relationnel dans le cas de la prise en compte de l'ascendant.

➢ Vous pouvez également vous intéresser aux **`Professionnels`**.
➢ …
Rappel : Pour effectuer vos requêtes d'utiliser des **`PreparedStatement`**

Annexe 1 : Mapping Objet/Relationnel tenant compte de l'ascendant

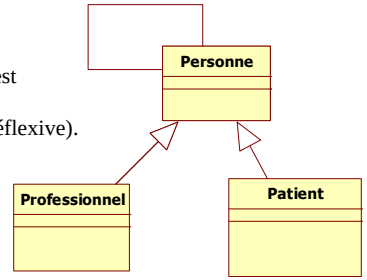
Rappel : Mapping objet/Relationnel sans ascendant :



La différence dans la base de données entre un enregistrement de type `Patient` et un enregistrement de type `Professionnel` se fait en examinant la valeur du champ `idTypePersonne` qui adopte la convention suivante :

- lorsque le champ `idTypePersonne` a pour valeur **`0`**, l'objet correspondant (à cet enregistrement) dans le programme Java est une instance de la classe `Patient`
- lorsque le champ `idTypePersonne` a pour valeur **`1`**, l'objet correspondant (à cet enregistrement) dans le programme Java est une instance de la classe `Professionnel`

Mapping objet/Relationnel AVEC ascendant :



Le diagramme de classes obtenu en tenant compte de l'ascendant est représenté ci-contre.
La classe `Personne` est désormais reliée à elle-même (relation réflexive).
Cette nouvelle relation est prise en compte dans notre mapping **Objet/Relationnel** par le champ `idTypePersonne` de la table `Personne.`
Le champ `idTypePersonne` de la table `Personne` pourra désormais prendre 3 valeurs :

- valeur **`0`** pour les instances de la classe **`Patient`**
- valeur **`1`** pour les instances de la classe **`Professionnel`**
- valeur **`2`** pour indiquer que la **`Personne a un Ascendant`**. Dans ce cas-là, le champ `idAscendant` devra également être renseigné puisque c'est lui qui permettra d'identifier l'ascendant (voir exemple avec DUPONT Toto dans la table `Personne` ci-dessous)

Personne

idPersonne	nom	prenom	datenaissan	male	telephone	portable	email	idAscendan	idTypePersc	nir	medecinTra	immatricula	specialite
1	DUPONT	Julie	21/05/1960	☐	0555434355	0606060606	julie.dupont@		0	2600587001123	LEDOC Paul		
2	LEDOC	Paul	10/07/1976	☑	0555434343	0612345678	paul.ledoc@le		1			871255358	généraliste
3	CHILDREN	Rose	16/02/1970	☐	0555434343	0678654321	rose.children@		1			312444555	pédiatrie
4	DURAND	Alfred	23/05/1968	☑	0512348989	0605050505	alfred.durant@		0	168072B12345(	LEDOC Paul		
5	DUPONT	Toto	25/12/1991	☑	0555430000	0605040302	toto.dupont@	1	2	2600587001123	LEDOC Paul		

Vous remarquez donc que les ***ascendants*** qui, dans notre application sont aussi des `Patient`(s), sont notés par leur ***propre type*** (`idTypePersonne =2`)
Ce nouveau mapping va bien sûr un peu compliquer le code.
Par exemple, lors de l'implémentation de la méthode `findAllPersonne`, il faudra dans un premier temps, **charger tous les patients sans ascendants** (c-a-d uniquement les enregistrements pour lesquels le champ `idTypePersonne` vaut `0`).
Ensuite, pour chaque `Patient` correctement "mappé" dans le programme Java, il faudra rechercher dans la base s'il possède des *descendants* (c-a-d si un enregistrement de type ascendant le référence : test sur `idTypePersonne` et `idAscendant`) . Ceci nous permettra alors de recréer correctement les objets du programme Java puisque rappelons-le l'attribut `unAscendant` de la classe `Personne` est de type `Personne` : on doit donc passer à l'attribut `unAscendant` une référence de type `Personne` (c-a-d que l'objet référençant l'ascendant doit être créé au préalable pour pouvoir passer sa référence …)