INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

CELINE CHAUGNY
DAMIEN POINTIN

Plateforme d'analyse technique pour la finance

Projet Fin d'Etude

Table des matières

1 Introduction **2**
1.1 Introduction . . . 2

2 Techniques utilisées **3**
2.1 Introduction . . . 3
2.2 Communiquer avec la base de données . . . 3
2.2.1 Connecteur JDBC . . . 3
2.2.2 DAO . . . 4
2.2.3 La sérialisation . . . 5
2.2.4 Notre choix : DAO vs Serialisation . . . 6
2.3 Introduction . . . 6
2.4 Cryptage du mot de passe . . . 6
2.4.1 Fonctions SQL . . . 7
2.4.2 Jasypt . . . 7
2.4.3 Notre choix . . . 8
2.5 Introduction . . . 8
2.6 Introduction . . . 8

3 Théorie Finance **9**
3.1 Introduction . . . 9
3.2 Introduction . . . 9
3.3 Introduction . . . 9
3.4 Introduction . . . 9

4 Modélisation **10**
4.1 Introduction . . . 10
4.2 Introduction . . . 10
4.3 Introduction . . . 10
4.4 Introduction . . . 10
4.5 Introduction . . . 10
4.6 Introduction . . . 10

5 Utilisation de l'application **11**
5.1 Introduction . . . 11

6 Gestion de projet **12**
6.1 Introduction . . . 12
6.2 Introduction . . . 12

7 Conclusion **13**
7.1 Introduction . . . 13

8 Bibliographie **14**

Chapitre 1
Introduction

1.1 Introduction

Chapitre 2
Techniques utilisées

2.1 Introduction

2.2 Communiquer avec la base de données

2.2.1 Connecteur JDBC

JDBC signifie Java Data Base Connectivity. C'est une interface Java qui permet d'accéder à une base de données SQL. L'avantage de ce driver est qu'il permet de se connecter à une base, d'exécuter des requêtes et d'en récupérer les résultats de manière simple. Tout se fait en Java, ce qui rend l'utilisation simple. Il suffit de connaître les commandes SQL. Une fois que le pilote est chargé dans le code Java, on se connecte à la base de données voulue avec l'URL, l'utilisateur et le mot-de-passe. Le package java.sql est ici nécessaire. Lorsque la connexion a été établie, l'exécution d'une requête se fait de la manière suivante :

```
// Chargement du pilote :
Class.forName("com.mysql.jdbc.Driver").newInstance();

// Connexion a la base de donnees :
java.sql.Connection connexion = DriverManager.getConnection("jdbc:mysql:URL", "utilisateur", "
    motDePasse");

// Generation de l'objet statement associe  la connexion :
java.sql.Statement statement = connexion.createStatement();
```

Une fois la connexion établie, il est possible d'interagir avec la base de données. Dans les exemples ci-dessous, nous supposerons que la base de données est composée d'une table Joueur dont les attributs sont des chaînes de caractères représentant le nom et le prénom des joueurs. Il existe deux fonctions principales associées à différents types de requêtes et qui s'appliquent à l'objet Statement :

— executeQuery : permet d'exécuter une requête du type 'SELECT' par exemple et qui donne retourne certaines lignes d'une table.

```
// Ecriture et execution de la requete SQL :
String requete = "SELECT * FROM Joueur";
ResultSet resultat = statement.executeQuery(requete);

```

— executeUpdate : permet de mettre à jour une table de la base de données dans le cadre de requêtes du type 'UPDATE', 'INSERT', 'DELETE'.

```
// Mise a jour d'une table (ajout d'un joueur) :
String udpate = "INSERT INTO Joueur (nom, prenom) VALUES ('Dupont', 'Jean')";
int nbLignesMAJ = statement.executeUpdate(update);
```

Une fois que la requête a été exécutée, il ne reste plus qu'à lire les données reçues. Dans le cadre de l'utilisation de la fonction executeUpdate, on récupère simplement un entier qui donne le nombre de lignes mise à jour dans la table. En revanche, la fonction executeQuery retourne un objet du type ResultSet qui est moins trivial à étudier. Cet objet contient une certain nombre de lignes répondant aux conditions de la requête et peut être comparé à une table d'une base de données. Il est composé de colonnes symbolisant les attributs des enregistrements de la table : ils possèdent un nom et un domaine dans lequel ils prennent leur valeur (int, float, char, ...). De plus, pour parcourir cet objet on a une sorte de curseurs qui point sur une ligne. On utilise la commande next pour passer à la ligne suivante si elle existe.

```
// Parcours du ResultSet s'il est non nul et tant qu'il y a une ligne :
Vector<String> noms;
Vector<String> prenoms;
if (resultat != null) {
  while (resultat.next()) {
    noms.add(resultat.getString("nom"));
    prenoms.add(resultat.getString("prenom"));
  }
}
```

Après l'exécution de requêtes et la récupération des données, il ne reste plus qu'à 'refermer' la connexion de la manière suivante :

```
// Fermeture des ojets :
statement.close();
connexion.close();
```

Les exemples de codes précédents ne prennent pas en compte les erreurs pouvant être générées lors de la compilation de ce code. En réalité, il faut protéger les instructions et lever les exceptions générées. Erreurs possibles :

- Lors de la connexion : échec de la connexion (ex : mauvais mot de passe, URL invalide,...).
- Lors de l'exécution d'une requête : mauvaise syntaxe ou appel de table/attributs inexistants.
- Lors de la déconnexion.

2.2.2 DAO

Présentation

Un objet d'accès aux données (Data Access Object ou DAO en anglais) est un modèle qui permet d'isoler les méthodes concernant le stockage des données et de ne pas les écrire directement dans nos classes métier. Ainsi le modèle DAO permet de regrouper l'accès aux données dans des classes à part plutôt que de les disperser.

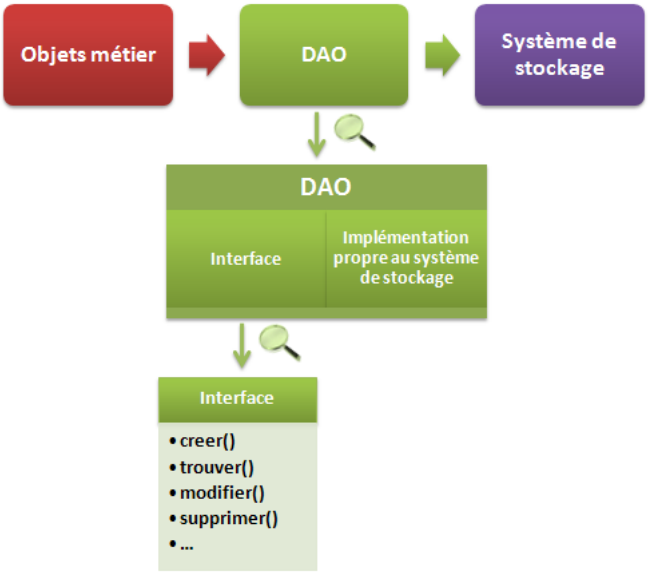
Le but du modèle DAO est d'arriver à encapsuler les méthodes concernant la communication avec la base de données dans une coucher pour arriver au schéma suivant :

Mise en place

La couche DAO va gérer les opérations classiques de stockage : ajout, lecture, modification et suppression. Ces quatre opérations sont souvent raccourcis par l'acronyme CRUD (create, read, update and delete).

Pour mettre en application le modèle DAO nous avons du créer nos propres exceptions. En effet, il faut que les exceptions générées par SQL ou JDBC soit référencées comme étant des exceptions dues à la boite noire DAO.

En reprenant la même idée, DAO se propose d'offrir une interface pour chacun des objets décrivant l'ensemble des méthodes qui seront accessibles dans l'objet. Ainsi, peu importe l'implémentation effectuée on pourra connaître les diverses méthodes de nos classes. Les interfaces permettent de décrire les méthodes des objets la couche donnée et ce n'est que l'implémentation qui sera dépendante du mode de stockage. Par exemple, pour un stockage SQL nous pouvons utiliser le driver JDBC présenté ci-dessus.



La couche DAO comportera également une fabrique. Cette fabrique sera unique et ne sera instanciée que si les informations de configuration sont correctes. Le but de cette fabrique sera de fournir une instance des différentes implémentations de la DAO.

Avantage et Inconvénient

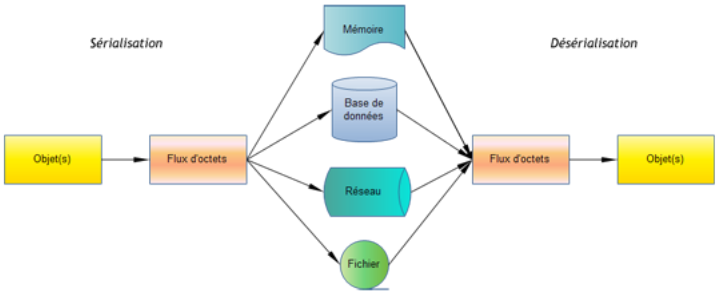
L'avantage du modèle DAO est donc clair, le changement du stockage du mode de stockage de données est simple. En effet, nous aurons seulement à changer nos classes DAO pour changer ce mode de stockage. L'inconvénient est du à la mise en œuvre qui demande une couche supplémentaire.

2.2.3 La sérialisation

Principe

La sérialisation est le processus de conversion d'un objet pour l'enregistrer dans une base de données ou un fichier par exemple. Le processus inverse s'appelle la désérialisation. Nous pouvons donc représenter

cette méthode selon le schéma suivant.



Mise en place

On peut choisir par exemple d'effectuer sa sérialisation en XML.

La classe XMLEncoder permet de sérialiser un objet en XML. Cette sérialisation ne prend en compte que les champs ayant un getter et un setter public. XMLEncoder(output) crée une nouvelle instance qui utilise le flux passé en paramètre comme résultat de la sérialisation.

La classe XMLDecoder permet quand à elle de désérialiser un objet à partir d'un document XML.

Pour notre application nous aurions besoin d'effectuer une sérialisation vers SQL. Prenons l'exemple du joueur, pour effectuer la sérialisation nous avons besoin d'une classe Joueur qui implémente Serializable, d'avoir un constructeur par défaut et d'avoir un getter et un setter public pour chaque attribut. Ensuite notre table SQL java doit contenir un id et un longblob (Binary Large Object). Ensuite nous n'avons plus qu'a effectue une sauvegarde dans la base de données. Pour la desérialisation, il nous faut récupérer le longblob pour le convertir en ObjectInputStream pour utiliser la méthode readObject().

Avantage et Inconvénient

La sérialisation présente des avantages comme une bonne portabilité ou le fait que le processus de sérialisation ne prenne pas en compte les champs qui ont leur valeur par défaut. Elle présente les inconvénients suivants :

— ne peut s'utiliser que sur des objets respectant la convention JavaBeans (constructeur par défaut, getter et setter pour tous les attributs ..)
— la taille des données sérialisés est plus importante que leur équivalent binaire

2.2.4 Notre choix : DAO vs Serialisation

Pour la communication avec notre base de données nous avons choisi d'utiliser le modèle DAO.

Le modèle DAO demande plus de travail pour être mis en place que la sérialisation mais possède l'avantage de nous fournir une base de données plus facilement utilisable dans un autre contexte. En effet, chacune de nos tables ne seront pas simplement un blob mais une valeur pour chacun des champs de notre table.

2.3 Introduction

2.4 Cryptage du mot de passe

Lors du déroulement du jeu nous allons avoir besoin de stocker le mot de passe du joueur. Pour cela, nous avons pensé qu'il était préférable d'encrypter ce mot de passe lors de son stockage.
Nous avons ainsi chercher différentes méthodes pour effectuer l'encryptage du mot de passe.

2.4.1 Fonctions SQL

Dans le langage SQL, la fonction MDA5() permet de chiffrer une chaîne de caractère en un entier hexadécimal de 32 caractères.
L'algorithme MDA() est une fonction de hachage cryptographique qui calcule à partir d'une chaîne de caractère son empreinte avec une probabilité très forte que deux empreintes soient différentes. Depuis 2004, une équipe chinoise a découvert des collisions complètes et MD5 n'est donc plus considéré comme sur au sens cryptographique.

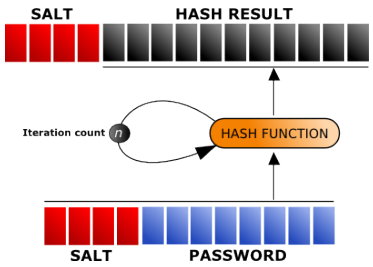
La fonction SQL SHA1() permet de chiffrer une chaîne de caractère sous la forme d'un chaîne de caractères de 40 caractères. SHA1 est également une fonction de hachage cryptographique. Elle a l'avantage d'être considéré comme sur contrairement à MDA().

2.4.2 Jasypt

Jasypt est une librairie java qui permet d'encrypter facilement les mots de passe avec une une grande sécurité.
Jasypt possede les avantages suivants :

- il permet de choisir la fonction de hachage que nous souhaitons (MDA ou SHA par exemple)
- il ajoute un salage au mot de passe qui permet d'avoir deux mots de passe cryptés différents pour le même mot de passe de départ
- il applique un nombre aléatoire de fois notre fonction de hachage (nombre > 1000 pour rendre plus difficile les attaques)

Nous pouvons résumer l'algorithme de chiffrement de Jasypt comme suit :



Le code pour encrypter le mot de passe est très simple, il nous suffit de créer un objet de type ConfigurablePasswordEncryptor, de définir l'algorithme de chiffrement. La méthode setPlainDigest nous permet avec l'argument false de choisir la méthode la plus sure avec un salage et un nombre d'itération aléatoire pour notre fonction de hachage. Enfin il ne nous reste plus qu'a appeler la méthode encryptPassword qui nous renvoie notre mot de passe encrypté à partir d'un mot de passe donné en entrée.

```
1 ConfigurablePasswordEncryptor passwordEncryptor = new ConfigurablePasswordEncryptor();
  passwordEncryptor.setAlgorithm( ALGO_CHIFFREMENT );
3 passwordEncryptor.setPlainDigest( false );
  String motDePasseChiffre = passwordEncryptor.encryptPassword( motDePasse );
```

De même que pour vérifier que notre mot de passe correspond au mot de passe chiffré il existe une méthode qui nous renvoie vraie en cas de correspondance :

```
passwordEncryptor.checkPassword(motDePasse, motDePasseChiffre )
```

2.4.3 Notre choix

L'inconvénient d'utiliser les fonctions de SQL est que l'on choisi une manière de crypter dépendante de notre base de données. Si nous décidons de changer notre manière de stocker notre base de données nous devrons ainsi trouver une nouvelle fonction.

Nous avons donc choisi d'utiliser Jasypt pour encrypter notre mot de passe. Cette librairie a l'avantage de nous permettre d'encrypter uen chaîne de caractère de manière relativement sure sans avoir de grandes compétences en cryptographie. En effet, nous ne connaissons pas en détail le fonctionnement de l'algorithme de cryptage mais avons simplement une idée globale de son fonctionnement.

Nous avons choisi comme fonction de hachage (ALGOCHIFFREMENT) SHA car nous avons que MDA n'est plus sur. Une fois l'encryptage du mot de passe effectué nous obtenons une chaîne de caractères de taille 56.

2.5 Introduction

2.6 Introduction

Chapitre 3
Théorie Finance

3.1 Introduction

3.2 Introduction

3.3 Introduction

3.4 Introduction

Chapitre 4
Modélisation

4.1 Introduction

4.2 Introduction

4.3 Introduction

4.4 Introduction

4.5 Introduction

4.6 Introduction

Chapitre 5
Utilisation de l'application

5.1 Introduction

Chapitre 6
Gestion de projet

6.1 Introduction

6.2 Introduction

Chapitre 7
Conclusion

7.1 Introduction

Chapitre 8
Bibliographie

http ://www.jasypt.org/howtoencryptuserpasswords.html
http ://sql.sh/fonctions/sha1
http ://sql.sh/fonctions/md5
https ://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee/le-modele-dao
http ://www.easywayserver.com/blog/save-serializable-object-in-java/

RENSEIGNEMENTS

Département GM
02.32.95.65.31
gm@insa-rouen.fr

INSA Rouen
Campus du Madrillet
685 avenue de l'Université – BP 08
76801 SAINT-ÉTIENNE-DU-ROUVRAY cedex
www.insa-rouen.fr

Membre de

GROUPE INSA
Normandie Université

Financeurs institutionnels

RÉGION HAUTE
MINISTÈRE DE L'ÉDUCATION NATIONALE, DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE