Personnel

Berteaux, Fourti, Meziani | 04/04/2023

Présentation

Un des responsables de la M2L, utilise une application pour gérer les employés des ligues.

Cette application, très simple, n'existe qu'en ligne de commande et est mono-utilisateur. Nous souhaiterions désigner un administrateur par ligue et lui confier la tâche de recenser les employés de sa ligue. Une partie du travail est déjà faite mais vous allez devoir le compléter.

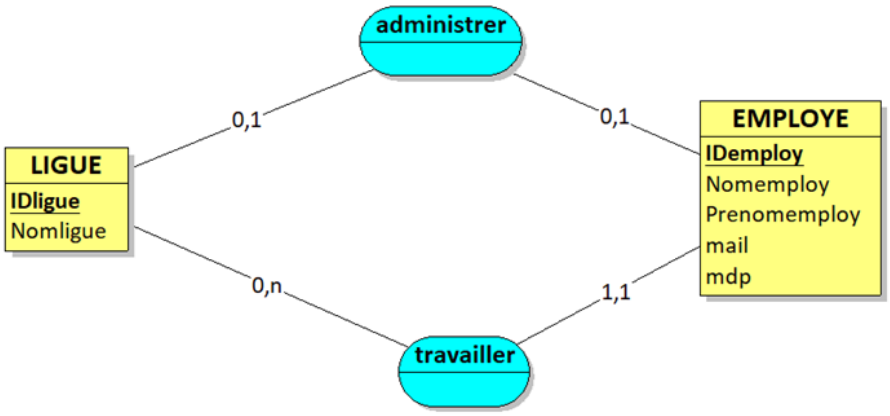
Spécification du besoin

Les niveaux d'habilitation des utilisateurs sont les suivants :

- Un simple employé de ligue peut ouvrir l'application et s'en servir comme un annuaire, mais il ne dispose d'aucun droit d'écriture.
- Un employé par ligue est administrateur et dispose de droits d'écriture peut gérer la liste des employés de sa propre ligue avec une application bureau.
- Le super-administrateur a accès en écriture à tous les employés des ligues. Il peut aussi gérer les comptes des administrateurs des ligues avec une application accessible en ligne de commande.
- L'application doit être rendue multi-utilisateurs grâce à l'utilisation d'une base de données.
- Les trois niveaux d'habilitation ci-dessus doivent être mis en place.

ITERATION 1

- Modélisation d'une base de données avec un MCD.



- Vérification du fonctionnement correct de l'application grâce à des tests unitaires.

Employe.java

```java
/**
 * Retourne le nom de l'employé.
 * @return le nom de l'employé.
 */

public String getNom()
{
    return nom;
}

/**
 * Change le nom de l'employé.
 * @param nom le nouveau nom.
 */

public void setNom(String nom)
{
    this.nom = nom;
    try {
        this.UpdateEmploye("nom_employe");
    } catch (SauvegardeImpossible e) {
        e.printStackTrace();
    }
}
```

Employe.jave nous a été donné, il était déjà bien rempli quand on a commencé à le travailler.

Pour cette mission, nous devions tester les différentes méthodes présente dans les fichiers qui nous avait été donné.

J'ai pris l'exemple de getNom et setNom car ils ne sont pas très compliqués à expliquer. Nous devions juste montrer que getNom affichait bien le nom et que SetNom permettait de le modifier.

« assertEqual » permet de vérifier que le paramètre 2 et égale au paramètre 1.

TestEmploye.java

```java
@Test
void GetNom () throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",LocalDate.now(),(LocalDate.of(2023, 12, 20)));
    assertEquals("Bouchard", employe.getNom());
}

@Test
void SetNom () throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",LocalDate.now(),(LocalDate.of(2023, 12, 20)));
    employe.setNom("Boulanger");
    assertEquals("Boulanger", employe.getNom());
}
```

- Gestion de la date de départ et de celle d'arrivée de chaque employé (couche métier + tests unitaires).

Employe.java

```java
/**
 * Retourne la date d'arrivée.
 * @return la date d'arrivée.
 */

public LocalDate getDateArrivee()
{
    return this.dateArrivee;
}

/**
 * Retourne la date de départ.
 * @return la date de départ.
 */

public LocalDate getDateDepart()
{
    return this.dateDepart;

}

/**
 * Change la date de départ.
 * @param date la nouvelle date de départ.
 */

public void setDateDepart(LocalDate dateDepart)
{
    this.dateDepart = dateDepart;
    try {
        this.UpdateEmploye("dateArrivee_employe");
    } catch (SauvegardeImpossible e) {
        e.printStackTrace();
    }
}
```

Comme dans l'exemple précédent, il fallait vérifier que getDateArrivee et getDateDepart affichait bien la date demandée.

Pour SetDateDepart, il fallait montrer qu'on pouvait changer la date.

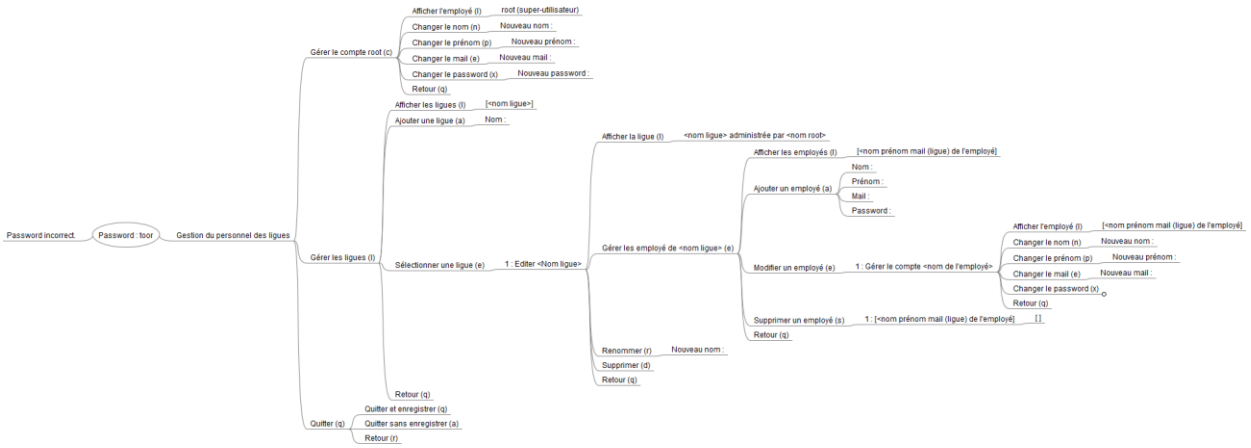
Mais pourquoi n'y a-t-il pas de SetDateArrivee ?

Je suis parti du principe que lorsqu'un employé arrive, la valeur de DateArrivee prend directement la date du jour où il est arrivé donc pas besoin de le modifier, il est arrivé ce jour-là, c'est un fait.

TestEmploye.java

```java
@Test
void GetDateArrivee ()throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",LocalDate.now(),(LocalDate.of(2023, 12, 20)));
    assertEquals(LocalDate.now(), employe.getDateArrivee());

}


@Test
void GetDateDepart () throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",LocalDate.now(),(LocalDate.of(2023, 12, 20)));
    assertEquals((LocalDate.of(2023, 12, 20)), employe.getDateDepart());

}

@Test
void SetDateDepart () throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",LocalDate.now(),(LocalDate.of(2023, 12, 20)));
    employe.setDateDepart((LocalDate.of(2023, 10, 10)));
    assertEquals((LocalDate.of(2023, 10, 10)), employe.getDateDepart());
}
```

- Représentation des menus du dialogue en ligne de commande avec un arbre heuristique (Utilisez un logiciel de type Freemind).



ITERATION 2

- Création de la base de données et production du script de création de tables.

```sql
DROP DATABASE IF EXISTS personnel;
CREATE DATABASE personnel;

USE personnel;

DROP TABLE IF EXISTS LIGUE;
CREATE TABLE LIGUE(
    num_ligue INT PRIMARY KEY AUTO_INCREMENT,
    nom_ligue VARCHAR(32)
);

DROP TABLE IF EXISTS EMPLOYE;
CREATE TABLE EMPLOYE(
    id_employe INT PRIMARY KEY  AUTO_INCREMENT,
    nom_employe VARCHAR(32) NOT NULL,
    prenom_employe VARCHAR(32) NOT NULL,
    mail_employe VARCHAR(32) NOT NULL,
    password_employe VARCHAR(32) NOT NULL,
    dateArrivee_employe DATE,
    dateDepart_employe DATE,
    admin INT DEFAULT 0,
    num_ligue INT NULL,
    FOREIGN KEY (num_ligue) REFERENCES LIGUE(num_ligue)
);
```

- Gestion des dates dans le dialogue en ligne de commande.

```
Option modifierEmploye(Employe employe)
{
    Menu menu = new Menu("Modifier le compte " + employe.getNom(), "m");
    menu.add(afficher(employe));
    menu.add(changerNom(employe));
    menu.add(changerPrenom(employe));
    menu.add(changerMail(employe));
    menu.add(changerPassword(employe));
    menu.add(changerDateDepart(employe));
    menu.addBack("q");
    return menu;
}
```

```
private Option changerDateDepart(final Employe employe)
{
    return new Option("Changer la date de départ", "d", () -> {employe.setDateDepart(LocalDate.parse(getString("Nouvelle date (YYYY-MM-DD) : ")));});
}
```

Dans l'exemple suivant, j'ai dû convertir LocalDate en String afin de pouvoir rentrer des valeurs qui seront accepté par l'application à l'aide de « .parse ».

```
private Option ajouterEmploye(final Ligue ligue)
{
    return new Option("ajouter un employé", "a",
            () ->
            {
                LocalDate.now();
                ligue.addEmploye(getString("nom : "),
                    getString("prenom : "), getString("mail : "),
                    getString("password : "),LocalDate.parse(getString("Date d'arrivée :")), LocalDate.parse(getString("Date de départ (YYYY-MM-DD) : ")));
            }
    );
}
```

- Dans le dialogue en ligne de commande, un employé doit être sélectionné avant que l'on puisse choisir de modifier ou de supprimer.

```
private Menu gererEmployes(Ligue ligue)
{
    Menu menu = new Menu("Gérer les employés de " + ligue.getNom(), "e");
    menu.add(afficherEmployes(ligue));
    menu.add(ajouterEmploye(ligue));
    menu.add(SelectionnerEmploye(ligue));
    menu.addBack("q");
    return menu;
}
```

```
private List<Employe> SelectionnerEmploye(final Ligue ligue)
{
    return new List<>("Sélectionner un employé", "s",
            () -> new ArrayList<>(ligue.getEmployes()),
            employeConsole.editerEmploye()
            );
}
```

- Possibilité de changer l'administrateur d'une ligue en ligne de commande.

```
private Menu editerLigue(Ligue ligue)
{
    Menu menu = new Menu("Editer " + ligue.getNom());
    menu.add(afficher(ligue));
    menu.add(gererEmployes(ligue));
    menu.add(changerAdministrateur(ligue));
    menu.add(changerNom(ligue));
    menu.add(supprimer(ligue));
    menu.addBack("q");
    return menu;
}
```

```
private List<Employe> changerAdministrateur(final Ligue ligue)
{
    return new List<>("Changer l'administrateur", "c",
            () -> new ArrayList<>(ligue.getEmployes()),
            (index, element) -> {ligue.setAdministrateur(element);}
            );
}
```

ITERATION 3

- Création d'une classe fille de Passerelle permettant de gérer le dialogue avec la base de données avec JDBC (ou avec Hibernate si vous le souhaitez).

```
public interface Passerelle
{
    public GestionPersonnel getGestionPersonnel();
    public void sauvegarderGestionPersonnel(GestionPersonnel gestionPersonnel)  throws SauvegardeImpossible;
    public int Insert(Ligue ligue) throws SauvegardeImpossible;
    public int Insert(Employe employe) throws SauvegardeImpossible;
    public void UpdateLigue(Ligue ligue) throws SauvegardeImpossible;
    public void UpdateEmploye(Employe employe, String string) throws SauvegardeImpossible;
    public void DeleteLigue(Ligue ligue) throws SauvegardeImpossible ;
    public void DeleteEmploye(Employe employe) throws SauvegardeImpossible;
}
```

- Utilisation de la base de données pour réaliser les opérations d'ajout, de modification, de suppression des ligues et des employés.

```
@Override
public int Insert(Employe employe) throws SauvegardeImpossible
{
    try {

        PreparedStatement instruction2;
        instruction2 = connection.prepareStatement("insert into employe (nom_employe, prenom_employe, mail_employe, password_employe, dateArrivee_employe, num_ligue) values(?,?,?,?,?,?)",
                Statement.RETURN_GENERATED_KEYS);
        instruction2.setString(1, employe.getNom());
        instruction2.setString(2, employe.getPrenom());
        instruction2.setString(3, employe.getMail());
        instruction2.setString(4, employe.getPassword());
        instruction2.setString(5, employe.getDateArrivee() == null ? null :  String.valueOf(employe.getDateArrivee()));
        instruction2.setInt(6, employe.getLigue().getId());
        instruction2.executeUpdate();
        ResultSet id = instruction2.getGeneratedKeys();
        id.next();
        return id.getInt(1);
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

```java
@Override
public void UpdateLigue(Ligue ligue) throws SauvegardeImpossible
{
    try
    {
        PreparedStatement instruction;
        instruction = connection.prepareStatement("UPDATE ligue SET nom_ligue = ? WHERE num_ligue = ?");
        instruction.setString(1, ligue.getNom());
        instruction.setInt(2, ligue.getId());
        instruction.executeUpdate();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        throw new SauvegardeImpossible(e);
    }
}
```

```java
@Override
public void DeleteLigue(Ligue ligue) throws SauvegardeImpossible
{
    try
    {
        PreparedStatement listLigue;
        listLigue = connection.prepareStatement("DELETE FROM ligue WHERE num_ligue = ?");
        listLigue.setInt(1, ligue.getId());
        listLigue.executeUpdate();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        throw new SauvegardeImpossible(e);
    }

}
```

- Modélisation de l'interface graphique avec des maquettes.