Documentation Technique
Système de Gestion du Personnel

TAMANI Yassine
LABIDI Souheyl
ARIAS Pablo

23 mars 2025

Table des matières

Chapitre 1

Introduction

1.1 Présentation du Projet

Le projet "Système de Gestion du Personnel" est une application Java destinée à gérer le personnel des ligues sportives hébergées par la Maison des Ligues (M2L). Ce système permet la gestion complète des ligues et de leurs employés, avec une structuration hiérarchique des droits d'accès et des responsabilités administratives.

Le système s'articule autour de plusieurs entités principales (ligues, employés) et intègre une gestion de rôles permettant de définir différents niveaux d'accès et de privilèges. Il offre également un système complet de persistance des données via JDBC et/ou sérialisation, selon la configuration choisie.

1.2 Objectifs du Projet

Le système vise à répondre aux besoins suivants :

— Gérer les ligues sportives (création, modification, suppression)

— Gérer les employés pour chaque ligue (ajout, modification, suppression)

— Gérer les dates d'arrivée et de départ des employés

— Attribuer et gérer des rôles (employé, administrateur de ligue, super-administrateur)

— Assurer la persistance des données

— Fournir une interface utilisateur en ligne de commande pour toutes les opérations

— Garantir la sécurité et la cohérence des données

1.3 Technologies Utilisées

Le projet est développé avec les technologies suivantes :

— **Langage de programmation** : Java 8

— **Gestion des dépendances** : Maven

— **Persistance des données** : JDBC avec MySQL, Sérialisation Java

— **Interface utilisateur** : CommandLineMenus (bibliothèque externe)

— **Tests** : JUnit 5

Chapitre 2

Architecture du Système

2.1 Vue d'Ensemble

L'architecture du système est basée sur le modèle MVC (Modèle-Vue-Contrôleur) simplifié :

- **Modèle** : Les classes du package personnel (GestionPersonnel, Ligue, Employe, Role)
- **Vue** : Les classes du package commandLine (PersonnelConsole, LigueConsole, EmployeConsole)
- **Contrôleur** : Les interactions entre les classes de modèle et les classes de vue

Le système utilise également le pattern Strategy pour la persistance des données, permettant de choisir entre la sérialisation et JDBC.

2.2 Organisation des Packages

Le projet est organisé en plusieurs packages, chacun ayant une responsabilité spécifique :

Package	Description
`personnel`	Contient les classes principales du modèle de données (GestionPersonnel, Ligue, Employe, Role) et les exceptions métier.
`jdbc`	Contient l'implémentation JDBC de l'interface Passerelle pour la persistance des données dans une base de données MySQL.
`serialisation`	Contient l'implémentation de sérialisation de l'interface Passerelle pour la persistance des données dans un fichier.
`commandLine`	Contient les classes de l'interface utilisateur en ligne de commande.
`testsUnitaires`	Contient les tests unitaires du système.

2.3 Diagramme de Classe Détaillé

Les principales classes du système et leurs relations sont détaillées ci-dessous :

2.3.1 Classe GestionPersonnel

GestionPersonnel

— **Attributs** :

— `private static GestionPersonnel gestionPersonnel` : Instance unique (singleton)

— `private SortedSet<Ligue> ligues` : Ensemble des ligues

— `private Employe root` : Super-administrateur

— `public final static int SERIALIZATION, JDBC, TYPE_PASSERELLE` : Constantes de configuration

— `private static Passerelle passerelle` : Instance de la passerelle de persistance

— **Méthodes principales** :

— `public static GestionPersonnel getGestionPersonnel()` : Méthode singleton

— `public void sauvegarder()` : Sauvegarde l'état du système

— `public Ligue addLigue(String nom)` : Ajoute une nouvelle ligue

— `public Employe getRoot()` : Retourne le super-administrateur

— `public Employe addRoot(String nom, String password)` : Crée un super-administrateur

2.3.2 Classe Ligue

Ligue

- **Attributs** :
  - `private int id` : Identifiant unique
  - `private String nom` : Nom de la ligue
  - `private SortedSet<Employe> employes` : Employés de la ligue
  - `private Employe administrateur` : Administrateur de la ligue
  - `private GestionPersonnel gestionPersonnel` : Référence à l'instance GestionPersonnel
- **Méthodes principales** :
  - `public String getNom()` : Retourne le nom de la ligue
  - `public void setNom(String nom)` : Modifie le nom de la ligue
  - `public Employe getAdministrateur()` : Retourne l'administrateur de la ligue
  - `public void setAdministrateur(Employe administrateur)` : Définit l'administrateur
  - `public SortedSet<Employe> getEmployes()` : Retourne les employés de la ligue
  - `public Employe addEmploye(...)` : Ajoute un employé à la ligue
  - `public void remove()` : Supprime la ligue et tous ses employés

2.3.3 Classe Employe

Employe

- **Attributs** :
  - `private int id` : Identifiant unique
  - `private String nom, prenom, password, mail` : Informations personnelles
  - `private Ligue ligue` : Ligue de rattachement
  - `private GestionPersonnel gestionPersonnel` : Référence à l'instance GestionPersonnel
  - `private LocalDate dateArrive, dateDepart` : Dates d'arrivée et de départ
  - `private int idRole` : Rôle de l'employé
- **Méthodes principales** :
  - Accesseurs et mutateurs pour tous les attributs avec validation des données
  - `public boolean estAdmin(Ligue ligue)` : Vérifie si l'employé est admin de la ligue
  - `public boolean estRoot()` : Vérifie si l'employé est le super-admin
  - `public void remove()` : Supprime l'employé (avec vérifications)
  - `public int compareTo(Employe autre)` : Implémentation de Comparable

2.3.4 Classe Role

Role

- **Attributs** :
  - `private int id` : Identifiant du rôle
  - `private String nom` : Nom du rôle
  - `public static final int SUPER_ADMIN, ADMIN_LIGUE, EMPLOYE` : Constantes des rôles
- **Méthodes** :
  - Accesseurs et mutateurs pour tous les attributs
  - `public String toString()` : Retourne le nom du rôle

2.3.5 Interface Passerelle

Passerelle

- **Méthodes** :
  - `public GestionPersonnel getGestionPersonnel()` : Charge les données
  - `public void sauvegarderGestionPersonnel(...)` : Sauvegarde les données
  - `public int insert(Ligue ligue)` : Insère une ligue
  - `public int insert(Employe employe)` : Insère un employé
  - `public void update(Ligue ligue)` : Met à jour une ligue
  - `public void update(Employe employe)` : Met à jour un employé
  - `public void delete(Employe employe)` : Supprime un employé
  - `public void delete(Ligue ligue)` : Supprime une ligue

Chapitre 3

Modèle de Données Détaillé

3.1 Classe GestionPersonnel

La classe `GestionPersonnel` est le point d'entrée principal du système et représente un singleton. Elle gère l'ensemble des ligues et fournit un accès au super-administrateur (root).

3.1.1 Attributs

- `private static final long serialVersionUID` : Nécessaire pour la sérialisation
- `private static GestionPersonnel gestionPersonnel` : Instance unique (pattern Singleton)
- `private SortedSet<Ligue> ligues` : Collection triée des ligues
- `private Employe root` : Référence au super-administrateur
- `public final static int SERIALIZATION = 1, JDBC = 2` : Constantes pour le choix du mode de persistance
- `public final static int TYPE_PASSERELLE = JDBC` : Mode de persistance actuel
- `private static Passerelle passerelle` : Instance de la passerelle vers le stockage des données

3.1.2 Constructeurs

- `public GestionPersonnel()` : Constructeur privé pour le singleton, initialise les attributs

3.1.3 Méthodes Principales

- `public static GestionPersonnel getGestionPersonnel()` : Méthode principale du pattern Singleton qui retourne l'instance unique. Si elle n'existe pas, elle la crée en chargeant depuis la passerelle ou en créant une nouvelle instance avec un super-administrateur par défaut.
- `public void sauvegarder()` : Sauvegarde l'état actuel du système via la passerelle configurée.

— `public Ligue getLigue(Employe administrateur)` : Retourne la ligue dont l'employé spécifié est l'administrateur, ou null s'il n'est administrateur d'aucune ligue.

— `public SortedSet<Ligue> getLigues()` : Retourne une vue non modifiable de l'ensemble des ligues.

— `public Ligue addLigue(String nom)` : Crée une nouvelle ligue avec le nom spécifié, l'insère dans la base de données, et la retourne.

— `public Ligue addLigue(int id, String nom)` : Surcharge utilisée lors du chargement depuis la base de données pour recréer des ligues existantes.

— `public Employe getRoot()` : Retourne le super-administrateur du système.

— `public Employe addRoot(String nom, String password)` : Crée un super-administrateur si aucun n'existe. Retourne le super-administrateur existant ou nouvellement créé.

— `public void addRoot(int id, String nom, String prenom, String mail, String password, LocalDate dateArrive, LocalDate dateDepart, int idRole)` : Méthode utilisée lors du chargement depuis la base de données pour recréer le super-administrateur.

3.1.4 Méthodes Utilitaires

— `void remove(Ligue ligue)` : Supprime une ligue de la collection interne (appelée après sa suppression de la base).

— `int insert(Ligue ligue)` : Insère une ligue dans la base via la passerelle et retourne son ID.

— `int insert(Employe employe)` : Insère un employé dans la base via la passerelle et retourne son ID.

— `void update(Ligue ligue)` : Met à jour une ligue dans la base via la passerelle.

— `void update(Employe employe)` : Met à jour un employé dans la base via la passerelle.

— `void delete(Employe employe)` : Supprime un employé de la base via la passerelle.

— `void delete(Ligue ligue)` : Supprime une ligue de la base via la passerelle.

3.2 Classe Ligue

La classe `Ligue` représente une ligue sportive avec ses employés et son administrateur.

3.2.1 Attributs

— `private static final long serialVersionUID` : Nécessaire pour la sérialisation

— `private int id` : Identifiant unique de la ligue (par défaut -1 jusqu'à insertion en base)

— `private String nom` : Nom de la ligue

— `private SortedSet<Employe> employes` : Collection triée des employés de la ligue

— `private Employe administrateur` : Référence à l'administrateur de la ligue

— `private GestionPersonnel gestionPersonnel` : Référence à l'instance de GestionPersonnel

3.2.2 Constructeurs

— `Ligue(GestionPersonnel gestionPersonnel, String nom)` : Crée une nouvelle ligue avec le nom spécifié, rattachée au GestionPersonnel.

— `Ligue(GestionPersonnel gestionPersonnel, int id, String nom)` : Crée une ligue avec un ID existant (utilisé lors du chargement depuis la base).

3.2.3 Méthodes Principales

— `public String getNom()` : Retourne le nom de la ligue.

— `public void setNom(String nom)` : Modifie le nom de la ligue et met à jour la base.

— `public Employe getAdministrateur()` : Retourne l'administrateur de la ligue.

— `public void setAdministrateur(Employe administrateur)` : Définit un nouvel administrateur pour la ligue. Vérifie que l'employé fait partie de la ligue ou est le root.

— `public SortedSet<Employe> getEmployes()` : Retourne la collection des employés.

— `public Employe addEmploye(String nom, String prenom, String mail, String password, LocalDate dateArv, LocalDate Datedeb)` : Crée un nouvel employé, l'ajoute à la ligue et le retourne.

— `public void remove()` : Supprime la ligue et tous ses employés de la base et du système.

— `public int compareTo(Ligue autre)` : Implémentation de l'interface Comparable, trie par nom.

— `public String toString()` : Représentation textuelle de la ligue.

— `public int getId()` et `public void setId(int id)` : Accesseurs pour l'identifiant.

3.2.4 Méthodes Utilitaires

— `void remove(Employe employe)` : Supprime un employé de la collection interne (appelé après sa suppression de la base).

3.3 Classe Employe

La classe `Employe` représente un employé d'une ligue avec ses informations personnelles, ses dates de contrat et son rôle.

3.3.1 Attributs

— `private static final long serialVersionUID` : Nécessaire pour la sérialisation

— `private int id` : Identifiant unique de l'employé

— `private String nom, prenom, password, mail` : Informations personnelles

— `private Ligue ligue` : Référence à la ligue de rattachement (null pour le root)

— `private GestionPersonnel gestionPersonnel` : Référence à l'instance de GestionPersonnel

— `private LocalDate dateArrive` : Date d'arrivée

— `private LocalDate dateDepart` : Date de départ prévue

— `private int idRole` : Identifiant du rôle de l'employé

3.3.2 Constructeurs

— `public Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrive, LocalDate dateDepart)` : Constructeur principal avec validation des dates.

— `Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password)` : Constructeur sans dates.

— `public Employe(GestionPersonnel gestionPersonnel, int id, String nom, String prenom, String mail, String password, LocalDate dateArrive, LocalDate dateDepart, Ligue ligue, int idRole)` : Constructeur complet pour le chargement depuis MySQL.

— `Employe(GestionPersonnel gestionPersonnel, String nom, String prenom, String mail, String password)` : Constructeur spécifique pour le root.

3.3.3 Méthodes de Gestion du Rôle

— `public boolean estAdmin(Ligue ligue)` : Vérifie si l'employé est l'administrateur de la ligue spécifiée.

— `public boolean estRoot()` : Vérifie si l'employé est le super-administrateur.

— `public boolean estAdmin()` : Vérifie si l'employé a le rôle d'administrateur de ligue.

— `public void setRole(int roleId)` : Modifie le rôle de l'employé.

— `public int getIdRole()` : Retourne l'identifiant du rôle.

3.3.4 Méthodes de Gestion des Dates

— `public LocalDate getDateArrivee()` : Retourne la date d'arrivée.

— `public void setDateArrivee(LocalDate dateArrive)` : Modifie la date d'arrivée avec validation.

— `public LocalDate getDateDepart()` : Retourne la date de départ.

— `public void setDateDepart(LocalDate dateDepart)` : Modifie la date de départ avec validation.

3.3.5 Accesseurs et Mutateurs

Pour tous les attributs (nom, prénom, mail, password, etc.) avec mise à jour de la base pour les mutateurs.

3.3.6 Autres Méthodes

— `public void remove()` : Supprime l'employé, avec vérification qu'il n'est pas le root.

— `public int compareTo(Employe autre)` : Implémentation de l'interface Comparable, trie par nom puis prénom.

— `public String toString()` : Représentation textuelle détaillée de l'employé.

3.4 Classe Role

La classe `Role` définit les différents rôles possibles dans le système.

3.4.1 Constantes

— `public static final int SUPER_ADMIN = 1` : Rôle de super-administrateur (root)

— `public static final int ADMIN_LIGUE = 2` : Rôle d'administrateur de ligue

— `public static final int EMPLOYE = 3` : Rôle d'employé régulier

3.4.2 Attributs

— `private int id` : Identifiant unique du rôle

— `private String nom` : Nom du rôle

3.4.3 Méthodes

— Accesseurs et mutateurs standard

— `public String toString()` : Retourne le nom du rôle

Chapitre 4

Persistance des Données

4.1 Architecture de Persistance

Le système utilise le pattern Strategy pour la persistance des données, avec deux implémentations possibles :

— JDBC : Stockage dans une base de données MySQL

— Sérialisation : Stockage dans un fichier binaire

La configuration se fait via la constante `TYPE_PASSERELLE` dans `GestionPersonnel`.

4.2 Interface Passerelle

L'interface `Passerelle` définit le contrat que doivent implémenter les classes de persistance.

— `public GestionPersonnel getGestionPersonnel()` : Charge toutes les données et retourne une instance complète.

— `public void sauvegarderGestionPersonnel(GestionPersonnel gestionPersonnel)` : Sauvegarde intégrale.

— `public int insert(Ligue ligue)` : Insère une ligue et retourne son ID.

— `public int insert(Employe employe)` : Insère un employé et retourne son ID.

— `public void update(Ligue ligue)` : Met à jour une ligue.

— `public void update(Employe employe)` : Met à jour un employé.

— `public void delete(Employe employe)` : Supprime un employé.

— `public void delete(Ligue ligue)` : Supprime une ligue.

4.3 Implémentation JDBC

La classe `JDBC` implémente l'interface `Passerelle` en utilisant JDBC pour stocker les données dans une base MySQL.

4.3.1 Attributs

— `Connection connection` : Connexion JDBC active

4.3.2 Méthodes Principales

- `public JDBC()` : Initialise la connexion à la base de données en utilisant les informations de `Credentials`.
- `public GestionPersonnel getGestionPersonnel()` : Charge toutes les données de la base :
  1. Charge le root (super-administrateur)
  2. Charge toutes les ligues
  3. Pour chaque ligue, charge ses employés
  4. Définit les administrateurs des ligues
- `public int insert(Ligue ligue)` : Insère une ligue dans la table `ligue`.
- `public int insert(Employe employe)` : Insère un employé dans la table `employe`, gère les transactions.
- `public void update(Ligue ligue)` : Met à jour une ligue dans la base.
- `public void update(Employe employe)` : Met à jour un employé dans la base, avec gestion des dates et des valeurs NULL.
- `public void delete(Employe employe)` : Supprime un employé de la base.
- `public void delete(Ligue ligue)` : Supprime une ligue et tous ses employés de la base, avec transactions.
- `public void close()` : Ferme la connexion à la base de données.

4.3.3 Structure de la Base de Données

Tables de la Base de Données

- **Table ligue** :
  - `ID_Ligue` : INT, PK, AUTO_INCREMENT
  - `nomLigue` : VARCHAR
  - `ID_Administrateur` : INT, FK vers employe.ID_Employe
- **Table employe** :
  - `ID_Employe` : INT, PK, AUTO_INCREMENT
  - `nomEmploye` : VARCHAR
  - `prenomEmploye` : VARCHAR
  - `mail` : VARCHAR
  - `passwd` : VARCHAR
  - `datearv` : DATE
  - `datedepart` : DATE
  - `ID_Ligue` : INT, FK vers ligue.ID_Ligue
  - `ID_Role` : INT

4.4 Implémentation Serialization

La classe `Serialization` implémente l'interface `Passerelle` en utilisant la sérialisation Java pour stocker les données dans un fichier.

4.4.1 Attributs

— `private static final String FILE_NAME = "personnel.serial"` : Nom du fichier de sauvegarde

4.4.2 Méthodes Principales

— `public GestionPersonnel getGestionPersonnel()` : Désérialise l'objet `GestionPersonnel` depuis le fichier.

— `public void sauvegarderGestionPersonnel(GestionPersonnel gestionPersonnel)` : Sérialise l'objet `GestionPersonnel` complet dans le fichier.

— Autres méthodes d'implémentation (insert, update, delete) qui retournent des valeurs par défaut car la vraie sauvegarde se fait en une seule fois.

Chapitre 5

Interface Utilisateur

5.1 Architecture de l'Interface

L'interface utilisateur est construite sur la bibliothèque CommandLineMenus qui permet de créer des menus hiérarchiques en ligne de commande. L'architecture est organisée en trois classes principales :

— `PersonnelConsole` : Point d'entrée principal et gestion de l'authentification

— `LigueConsole` : Gestion des ligues et de leurs employés

— `EmployeConsole` : Gestion des employés individuellement

5.2 Classe PersonnelConsole

La classe `PersonnelConsole` est le point d'entrée de l'application et gère l'authentification et le menu principal.

5.2.1 Attributs

— `private GestionPersonnel gestionPersonnel` : Référence au modèle

— `LigueConsole ligueConsole` : Référence à l'interface de gestion des ligues

— `EmployeConsole employeConsole` : Référence à l'interface de gestion des employés

5.2.2 Méthodes Principales

— `public PersonnelConsole(GestionPersonnel gestionPersonnel)` : Constructeur, initialise les références.

— `public void start()` : Démarre l'application en lançant le menu principal.

— `private Menu menuPrincipal()` : Crée et retourne le menu principal avec les options :

1. Éditer le profil du root
2. Gérer les ligues
3. Quitter

— `private Menu menuQuitter()` : Sous-menu pour quitter avec ou sans sauvegarde.

— `private boolean verifiePassword()` : Vérifie le mot de passe du super-administrateur pour l'authentification.

— `public static void main(String[] args)` : Point d'entrée de l'application, gère l'authentification initiale.

5.3 Classe LigueConsole

La classe `LigueConsole` gère l'interface utilisateur pour les opérations liées aux ligues.

5.3.1 Attributs

— `private GestionPersonnel gestionPersonnel` : Référence au modèle

— `private EmployeConsole employeConsole` : Référence à l'interface de gestion des employés

5.3.2 Méthodes de Menu

— `Menu menuLigues()` : Menu principal de gestion des ligues avec les options :

1. Afficher la liste des ligues
2. Ajouter une ligue
3. Sélectionner une ligue
4. Retour

— `private Menu editerLigue(Ligue ligue)` : Menu d'édition d'une ligue spécifique avec les options :

1. Afficher les informations de la ligue
2. Gérer les employés
3. Changer le nom de la ligue
4. Changer l'administrateur
5. Supprimer la ligue
6. Retour

— `private Menu gererEmployes(Ligue ligue)` : Menu de gestion des employés d'une ligue avec les options :

1. Afficher la liste des employés
2. Ajouter un employé
3. Gérer un employé spécifique
4. Retour

5.3.3 Méthodes d'Action

— `private Option afficherLigues()` : Affiche la liste des ligues existantes.

— `private Option afficher(final Ligue ligue)` : Affiche les détails d'une ligue.

— `private Option afficherEmployes(final Ligue ligue)` : Affiche la liste des employés d'une ligue.

— `private Option ajouterLigue()` : Ajoute une nouvelle ligue après saisie du nom.

— `private Option changerNom(final Ligue ligue)` : Change le nom d'une ligue existante.

— `private Option changerAdministrateurLigue(final Ligue ligue)` : Change l'administrateur d'une ligue en proposant une liste d'employés éligibles.

— `private Option ajouterEmploye(final Ligue ligue)` : Ajoute un nouvel employé à une ligue après saisie des informations complètes.

— `private Option supprimer(final Ligue ligue)` : Supprime une ligue après confirmation.

— `private Option supprimerEmploye(final Employe employe)` : Supprime un employé après confirmation.

5.3.4 Méthodes de Navigation

— `private List<Ligue> selectionnerLigue()` : Affiche une liste des ligues pour en sélectionner une.

— `private List<Employe> GererEmploye(final Ligue ligue)` : Affiche une liste des employés d'une ligue pour en sélectionner un.

— `private Menu gererEmploye(final Employe employe)` : Menu de gestion d'un employé spécifique.

— `private Option modifierEmploye(final Employe employe)` : Délègue la modification d'un employé à EmployeConsole.

5.4 Classe EmployeConsole

La classe `EmployeConsole` gère l'interface utilisateur pour les opérations liées aux employés individuels.

5.4.1 Méthodes de Menu

— `Menu editerEmploye(final Employe employe)` : Menu d'édition d'un employé avec les options :

1. Afficher les informations de l'employé
2. Modifier le nom
3. Modifier le prénom
4. Modifier le mail
5. Modifier les dates d'arrivée/départ
6. Modifier le mot de passe
7. Retour

5.4.2 Méthodes d'Action

— `private Option afficher(final Employe employe)` : Affiche les détails d'un employé.

— `private Option changerNom(final Employe employe)` : Change le nom d'un employé.

— `private Option changerPrenom(final Employe employe)` : Change le prénom d'un employé.

— `private Option changerMail(final Employe employe)` : Change l'email d'un employé.

— `private Option changerDateArrivee(final Employe employe)` : Change la date d'arrivée d'un employé.

— `private Option changerDateDepart(final Employe employe)` : Change la date de départ d'un employé.

— `private Option changerPassword(final Employe employe)` : Change le mot de passe d'un employé.

Chapitre 6

Gestion des Exceptions

6.1 Hiérarchie des Exceptions

Le système définit plusieurs exceptions spécifiques pour gérer les cas d'erreur du domaine métier :

Hiérarchie des Exceptions

- `SauvegardeImpossible` : Exception lors de la sauvegarde des données
  - Encapsule les exceptions sous-jacentes (SQLException, IOException)
- `Erreurdate` : Exception lors de la validation des dates
  - Lancée quand la date de départ est antérieure à la date d'arrivée
  - Lancée quand la date d'arrivée est null
- `ImpossibleDeSupprimerRoot` : Exception lors de la tentative de suppression du super-administrateur
- `DroitsInsuffisants` : Exception lors de tentative d'opération sans les droits nécessaires
  - Lancée notamment lors de la tentative de définir un administrateur qui n'est pas de la ligue

6.2 Gestion des Exceptions dans le Code

Les exceptions sont gérées de manière cohérente à travers le code :

- Au niveau du modèle (classes `GestionPersonnel`, `Ligue`, `Employe`) :
  - Vérification des préconditions et lancement des exceptions appropriées
  - Propagation des exceptions métier vers les couches supérieures
- Au niveau de la persistance (classes `JDBC`, `Serialization`) :
  - Capture des exceptions techniques (SQLException, IOException)
  - Encapsulation dans `SauvegardeImpossible` pour uniformiser la gestion
  - Journalisation des erreurs importantes
- Au niveau de l'interface (classes `*Console`) :
  - Capture et traitement final des exceptions

— Affichage de messages d'erreur adaptés à l'utilisateur
— Gestion des tentatives multiples (ex : saisie de dates)

Chapitre 7

Tests Unitaires

7.1 Architecture des Tests

Les tests unitaires sont organisés dans le package `testsUnitaires` et utilisent JUnit 5 (Jupiter). La classe `testLigue` contient l'ensemble des tests pour les ligues et les employés.

7.2 Tests des Ligues

- `void createLigue()` : Vérifie la création d'une ligue.
- `void setLigue()` : Vérifie la modification du nom d'une ligue.
- `void Suppression()` : Vérifie la suppression d'une ligue et de ses employés.

7.3 Tests des Administrateurs

- `void changementetSuppAdmin()` : Vérifie :
  - Le changement d'administrateur d'une ligue
  - La suppression d'un administrateur et le retour au root comme administrateur par défaut

7.4 Tests des Employés

- `void Employe()` : Vérifie les accesseurs et mutateurs des employés.
- `void addEmploye()` : Vérifie l'ajout d'un employé à une ligue.

7.5 Tests des Dates

- `void testValidDates()` : Vérifie que des dates valides sont acceptées.
- `void testInvalidDates()` : Vérifie que des dates invalides (départ avant arrivée) déclenchent une exception.
- `void testDateArriveNull()` : Vérifie qu'une date d'arrivée null déclenche une exception.

— `void setDateArriveNull()` : Vérifie qu'une tentative de définir une date d'arrivée null déclenche une exception.

— `void testSetDateDepartInvalid()` : Vérifie qu'une tentative de définir une date de départ invalide déclenche une exception.

— `void testSetDateArriveInvalid()` : Vérifie qu'une tentative de définir une date d'arrivée invalide déclenche une exception.

— `void testGetDate()` : Vérifie que les accesseurs des dates retournent les valeurs correctes.

Chapitre 8

Problèmes Identifiés et Améliorations Proposées

8.1 Problèmes Identifiés

Après analyse du code, plusieurs problèmes ont été identifiés :

8.1.1 Sécurité

— Stockage des mots de passe en clair sans hachage
— Absence de validation des entrées utilisateur (notamment les emails)
— Permissions et contrôle d'accès basiques

8.1.2 Architecture

— Redondance dans certains constructeurs d'Employe
— Inconsistances dans la gestion des dates (parfois nullable, parfois non)
— Couplage fort entre les classes du modèle et la persistance

8.1.3 Persistance

— Gestion des erreurs JDBC perfectible
— Problème avec le chargement du root depuis la base (retourne null au lieu de créer automatiquement)
— Paramètre idRole inutile dans addRoot

8.2 Améliorations Proposées

8.2.1 Architecture et Code

— Supprimer le constructeur redondant d'Employe (celui sans dates)
— Harmoniser la gestion des dates (toujours exiger des dates valides)
— Modifier addRoot() pour ne pas avoir de paramètre idRole (toujours SUPER_ADMIN)

— Appliquer le principe de séparation des préoccupations (SoC) entre modèle et persistance
— Utiliser des classes DTO (Data Transfer Objects) pour la persistance

8.2.2 Sécurité

— Implémenter le hachage des mots de passe (bcrypt, PBKDF2)
— Valider les entrées utilisateur (format d'email, longueur des champs, etc.)
— Améliorer le système de contrôle d'accès (droits plus granulaires)

8.2.3 Persistance

— Améliorer la gestion des erreurs JDBC (transactions plus robustes)
— Corriger le chargement du root pour créer automatiquement un root par défaut si aucun n'existe
— Optimiser les requêtes SQL (préparation des statements, pooling de connexions)

8.2.4 Interface Utilisateur

— Ajouter une interface graphique (Swing, JavaFX)
— Améliorer les retours utilisateur (messages plus clairs)
— Améliorer la gestion des erreurs de saisie
— Ajouter des fonctionnalités de recherche et filtrage

Chapitre 9

Documentation Technique Supplémentaire

9.1 Structure de la Base de Données

Description détaillée de la structure de la base de données MySQL utilisée par l'application.

Script de Création de la Base de Données

```
CREATE TABLE role (
    ID_Role INT PRIMARY KEY,
    nomRole VARCHAR(50) NOT NULL
);

CREATE TABLE ligue (
    ID_Ligue INT PRIMARY KEY AUTO_INCREMENT,
    nomLigue VARCHAR(100) NOT NULL,
    ID_Administrateur INT
);

CREATE TABLE employe (
    ID_Employe INT PRIMARY KEY AUTO_INCREMENT,
    nomEmploye VARCHAR(50) NOT NULL,
    prenomEmploye VARCHAR(50) NOT NULL,
    mail VARCHAR(100),
    passwd VARCHAR(50) NOT NULL,
    datearv DATE,
    datedepart DATE,
    ID_Ligue INT,
    ID_Role INT NOT NULL DEFAULT 3,
    FOREIGN KEY (ID_Ligue) REFERENCES ligue(ID_Ligue),
    FOREIGN KEY (ID_Role) REFERENCES role(ID_Role)
);

ALTER TABLE ligue
ADD CONSTRAINT fk_admin
FOREIGN KEY (ID_Administrateur)
REFERENCES employe(ID_Employe);

-- Insertion des rôles par défaut
INSERT INTO role VALUES (1, 'SUPER_ADMIN');
INSERT INTO role VALUES (2, 'ADMIN_LIGUE');
INSERT INTO role VALUES (3, 'EMPLOYE');
```

9.2 Configuration de la Connexion

La configuration de la connexion à la base de données se fait via la classe `Credentials` dans le package `jdbc`.

Configuration de la Connexion à la Base de Données

```
package jdbc;

/**
 * Informations de connexion à la base de données MySQL.
 */
public class Credentials {
    private static final String HOST = "localhost";
    private static final String PORT = "3306";
    private static final String DATABASE = "personnel";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    public static String getUrl() {
        return "jdbc:mysql://" + HOST + ":" + PORT + "/" + DATABASE +
               "?useSSL=false&useUnicode=true&useJDBCCompliantTimezoneShift=" +
               "true&useLegacyDatetimeCode=false&serverTimezone=UTC";
    }

    public static String getUser() {
        return USER;
    }

    public static String getPassword() {
        return PASSWORD;
    }

    public static String getDriverClassName() {
        return "com.mysql.cj.jdbc.Driver";
    }
}
```

9.3 Gestion des Dates

L'application utilise la classe `LocalDate` de Java 8 pour gérer les dates. Voici les règles métier concernant les dates :

— La date d'arrivée ne peut pas être null

— La date de départ ne peut pas être antérieure à la date d'arrivée

— Les dates sont validées à la création et à la modification d'un employé

— Les exceptions de type `Erreurdate` sont levées en cas de dates invalides

9.4 Cycle de Vie d'un Employé

1. Création uniquement via la méthode `addEmploye()` d'une ligue
2. Attribution automatique du rôle EMPLOYE

3. Possibilité de devenir administrateur de ligue
4. Suppression possible tant qu'il n'est pas le super-administrateur
5. Lors de la suppression d'un administrateur, le root devient l'administrateur par défaut

9.5 Hiérarchie des Rôles

1. **SUPER_ADMIN (1)** : Super-administrateur unique (root), non rattaché à une ligue
2. **ADMIN_LIGUE (2)** : Administrateur d'une ligue, a des droits étendus sur sa ligue
3. **EMPLOYE (3)** : Employé standard d'une ligue

9.6 Dépendances du Projet

Le projet utilise Maven pour la gestion des dépendances. Voici les principales dépendances :

— **MySQL Connector/J** (version 8.0.16) : Connecteur JDBC pour MySQL
— **CommandLineMenus** (version 2.0) : Bibliothèque pour la création des menus en ligne de commande
— **JUnit Jupiter** : Framework de test unitaire (implicite)

9.7 Compilation et Exécution

— **Compilation** : `mvn compile`
— **Tests** : `mvn test`
— **Package** : `mvn package`
— **Exécution** : `java -cp target/Personnel-2.0.jar commandLine.PersonnelConsole`

Chapitre 10

Conclusion

10.1 Récapitulatif

Le système de gestion du personnel pour les ligues sportives offre une solution complète pour gérer les ligues et leurs employés. Il propose :

- Une architecture orientée objet bien structurée
- Une gestion hiérarchique des utilisateurs et des droits
- Une persistance des données flexible (JDBC ou sérialisation)
- Une interface utilisateur en ligne de commande fonctionnelle
- Une validation des données saisies, notamment les dates
- Des tests unitaires couvrant les fonctionnalités principales

10.2 Points Forts

- Architecture modulaire et extensible
- Gestion claire des rôles et des droits
- Validation robuste des dates d'arrivée et de départ
- Interface utilisateur intuitive et bien structurée
- Transactions JDBC pour garantir l'intégrité des données

10.3 Limitations et Perspectives

- Sécurité à améliorer (hachage des mots de passe)
- Interface graphique à développer
- Fonctionnalités métier à enrichir (gestion des congés, planning)
- API REST pour l'intégration avec d'autres systèmes
- Déploiement en environnement réel à finaliser

10.4 Liens Utiles

- **Code source** : https://github.com/rabbyt3s/AP-Parking/tree/main/personnel-master
- **Documentation Java 8** : https://docs.oracle.com/javase/8/docs/api/