Rapport d'Implémentation Finale du Projet de Programmation Orientée Objet : Projet_POO_AV

Réalisé par :

MAKNI Youssef

2 BIS 2

Table des Matières

1. Introduction

Le projet "Projet_POO_AV" est une application de bureau développée en Java, conçue comme un gestionnaire de fichiers favori. Son objectif principal est de permettre à l'utilisateur de répertorier, d'organiser et de gérer une collection de fichiers importants en associant des métadonnées telles que le titre, l'auteur, des mots-clés (tags), un résumé et des commentaires.

L'application offre une interface graphique conviviale construite avec JavaFX et s'appuie sur une base de données Oracle pour la persistance des données. Ce rapport détaille l'implémentation finale du projet, en se concentrant sur la structure du code, les fonctionnalités offertes et les choix techniques opérés, tels qu'ils apparaissent dans le fichier source Projet_POO_AV.java. L'objectif réalisé par le code final est de fournir un outil fonctionnel et robuste pour la gestion personnalisée de références à des fichiers stockés localement.

2. Architecture Générale du Projet

L'application "Projet_POO_AV" adopte une architecture monolithique typique des applications de bureau JavaFX, intégrant la logique de présentation (GUI), la logique métier et l'accès aux données au sein d'une même structure de code.

2.1 Structure du Code Source

Le projet est contenu intégralement dans un unique fichier Projet_POO_AV.java. Ce fichier comprend :

- La classe principale Projet_POO_AV qui hérite de javafx.application.Application.
- Des classes internes : FavoriteFile, SearchCriteria, et TagCount.
- La gestion de la connexion à la base de données et les opérations CRUD via JDBC.
- La construction de l'interface utilisateur JavaFX (fenêtres, dialogues, tableau d'affichage).
- La logique métier pour chaque fonctionnalité.

3. Modèle de Données et Persistance

La persistance des informations sur les fichiers favoris est assurée par une base de

données Oracle.

3.1 Schéma de la Base de Données

Une seule table, nommée favorite_files, est utilisée pour stocker toutes les données. Sa structure est définie comme suit lors de la création (méthode createTableIfNotExists) :

```
CREATE TABLE favorite_files (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    file_path VARCHAR2(1000) NOT NULL UNIQUE,
    title VARCHAR2(255) NOT NULL,
    author VARCHAR2(255),
    tags VARCHAR2(1000) NOT NULL,
    summary CLOB,
    comments CLOB
);
```

- **id**: Identifiant numérique unique pour chaque enregistrement, auto-incrémenté (clé primaire).
- **file_path**: Chemin d'accès complet au fichier sur le système de l'utilisateur. Ce champ est obligatoire et unique pour éviter les doublons.
- **title**: Titre associé au fichier favori. Champ obligatoire.
- **author**: Auteur du fichier ou du contenu. Champ optionnel.
- **tags**: Mots-clés associés au fichier, séparés par des points-virgules. Champ obligatoire.
- **summary**: Résumé du contenu du fichier. Champ optionnel, de type CLOB pour stocker des textes longs.
- **comments**: Commentaires personnels de l'utilisateur sur le fichier. Champ optionnel, de type CLOB.

La méthode createTableIfNotExists tente de créer cette table au démarrage de l'application. Elle gère spécifiquement l'erreur ORA-00955 qui indique que la table existe déjà, permettant ainsi un fonctionnement normal si la table est déjà en place.

3.2 Configuration de la Connexion

Les paramètres de connexion à la base de données Oracle sont définis comme des constantes statiques dans la classe Projet_POO_AV:

- DB_URL: "jdbc:oracle:thin:@localhost:1521/FREEPDB1"

- DB_USER: "SYSTEM"
- DB_PASSWORD: "isg2025"

Le driver JDBC Oracle (oracle.jdbc.driver.OracleDriver) est chargé explicitement au démarrage de l'application. En cas d'échec du chargement du driver, l'application affiche un message d'erreur critique sur la console et se termine. La connexion est établie via la méthode getConnection(), qui utilise DriverManager.getConnection().

4. Fonctionnalités Implémentées

L'application "Projet_POO_AV" offre un ensemble de fonctionnalités pour la gestion des fichiers favoris.

4.1 Gestion des Fichiers Favoris (CRUD)

Les opérations fondamentales de création, lecture, mise à jour et suppression (CRUD) sont implémentées pour les fichiers favoris.

4.1.1 Ajout d'un Fichier Favori

- **Accès :** Bouton "Ajouter Favori".
- **Interface :** Une boîte de dialogue modale (showAddEditDialog(null)) s'ouvre, contenant un formulaire avec les champs suivants :
  - Chemin du fichier (non éditable directement, sélectionné via un bouton "Sélectionner...").
  - Titre (obligatoire).
  - Auteur.
  - Tags (obligatoires, séparés par des points-virgules).
  - Résumé.
  - Commentaires.
- **Logique :**
  - L'utilisateur sélectionne un fichier via un FileChooser. Le chemin absolu est affiché. Si le champ "Titre" est vide, il est automatiquement pré-rempli avec le nom du fichier sélectionné.
  - Validation des champs : le chemin du fichier, le titre et les tags sont obligatoires. Des alertes sont affichées en cas de non-respect.
  - Si la validation réussit, un nouvel objet FavoriteFile est créé et la méthode addFavoriteFile(FavoriteFile file) est appelée.
  - Cette méthode JDBC insère une nouvelle ligne dans la table favorite_files. Elle gère l'erreur ORA-00001 (violation de contrainte unique sur file_path) en affichant une alerte spécifique si le chemin du fichier existe déjà.
  - Après un ajout réussi, la liste des favoris dans l'interface est rafraîchie.

- **Code principal :** showAddEditDialog(), addFavoriteFile().

4.1.2 Modification d'un Fichier Favori

- **Accès :** Sélectionner un fichier dans le tableau puis cliquer sur "Modifier Sélection".
- **Interface :** La même boîte de dialogue que pour l'ajout (showAddEditDialog(selectedFile)) est utilisée, mais pré-remplie avec les informations du fichier sélectionné.
    - Le champ "Chemin du fichier" et le bouton "Sélectionner..." sont désactivés, car le chemin du fichier ne peut pas être modifié (il fait partie de l'identité unique du favori en base).
- **Logique :**
    - L'utilisateur modifie les champs souhaités (titre, auteur, tags, résumé, commentaires).
    - La validation des champs obligatoires (titre, tags) s'applique.
    - Si la validation réussit, la méthode updateFavoriteFile(FavoriteFile file) est appelée.
    - Cette méthode JDBC met à jour l'enregistrement correspondant dans la table favorite_files en se basant sur l'id du fichier.
    - Après une modification réussie, la liste des favoris est rafraîchie.
- **Code principal :** showAddEditDialog(), updateFavoriteFile().

4.1.3 Suppression d'un Fichier Favori

- **Accès :** Sélectionner un fichier dans le tableau puis cliquer sur "Supprimer Sélection".
- **Interface :** Une boîte de dialogue de confirmation (Alert.AlertType.CONFIRMATION) demande à l'utilisateur de valider la suppression.
- **Logique :**
    - Si l'utilisateur confirme, la méthode deleteFavoriteFile(int id) est appelée avec l'ID du fichier sélectionné.
    - Cette méthode JDBC supprime l'enregistrement correspondant de la table favorite_files.
    - Après une suppression réussie, la liste des favoris est rafraîchie.
- **Code principal :** deleteSelectedFile(), deleteFavoriteFile().

4.1.4 Affichage des Fichiers Favoris

- **Interface :** Un TableView (tableView) occupe la partie centrale de la fenêtre principale. Il affiche les fichiers favoris avec les colonnes suivantes :
    - Titre

    - Auteur
    - Tags
    - Chemin Fichier
    - Résumé
    - Commentaires
- **Logique :**
    - Au démarrage et après chaque opération CRUD ou recherche, la méthode loadFavoriteFilesAndUpdateTable() est appelée.
    - Elle invoque getAllFavoriteFiles() (ou searchFavoriteFiles() en cas de recherche) pour récupérer les données de la base.
    - Les données récupérées (une List<FavoriteFile>) sont ensuite utilisées pour mettre à jour l'ObservableList<FavoriteFile> (favoriteFilesData) liée au TableView, provoquant son rafraîchissement automatique.
    - Les fichiers sont initialement triés par titre de manière ascendante (défini dans la requête SQL de getAllFavoriteFiles).
- **Code principal :** setupTableView(), loadFavoriteFilesAndUpdateTable(), getAllFavoriteFiles().

4.2 Ouverture de Fichiers

- **Accès :** Double-clic sur une ligne (un fichier favori) dans le TableView.
- **Logique :**
    - L'événement de double-clic récupère l'objet FavoriteFile sélectionné.
    - Il vérifie si Desktop.isDesktopSupported() est vrai.
    - Si oui, il crée un objet java.io.File à partir du filePath stocké.
    - Si le fichier existe (fileToOpen.exists()), Desktop.getDesktop().open(fileToOpen) est appelé pour ouvrir le fichier avec l'application par défaut du système d'exploitation.
    - Des alertes d'erreur sont affichées si le fichier n'est pas trouvé ou si une IOException se produit lors de l'ouverture.
- **Code principal :** Listener setOnMouseClicked dans setupTableView().

4.3 Recherche de Fichiers Favoris

- **Accès :** Bouton "Rechercher".
- **Interface :** Une boîte de dialogue modale (showSearchDialog()) s'ouvre avec trois champs de texte pour saisir les critères de recherche :
    - Titre (contient...)
    - Auteur (contient...)
    - Tag (contient...)
- **Logique :**

    - L'utilisateur saisit les termes de recherche dans un ou plusieurs champs.
    - Lors du clic sur "Rechercher" dans la dialogue, un objet SearchCriteria est créé.
    - La méthode searchFavoriteFiles(SearchCriteria criteria) est appelée.
    - Cette méthode JDBC construit une requête SQL dynamique SELECT avec des clauses WHERE et LIKE pour chaque critère non vide. La recherche est insensible à la casse (utilisation de LOWER() en SQL et passage des paramètres en minuscules) et recherche des correspondances partielles (%term%).
    - Les résultats de la recherche (une List<FavoriteFile>) remplacent le contenu actuel du TableView.
    - Une alerte informe du nombre de résultats trouvés ou de l'absence de résultats.
- **Code principal :** showSearchDialog(), searchFavoriteFiles(), classe interne SearchCriteria.

4.4 Affichage des Propriétés

- **Accès :** Bouton "Propriétés".
- **Interface :** Une boîte de dialogue modale (showPropertiesDialog()) s'ouvre, affichant un TextArea non éditable avec des statistiques calculées sur l'ensemble des fichiers favoris stockés en base.
- **Logique :**
    - La méthode récupère tous les fichiers favoris via getAllFavoriteFiles().
    - Elle calcule et formate les informations suivantes :
        - Nombre total de fichiers favoris.
        - Liste des auteurs uniques (triée alphabétiquement).
        - Liste des tags uniques (appelés "Centres d'intérêt", triée alphabétiquement).
        - Nombre de fichiers par tag (trié par nombre d'occurrences décroissant, puis par nom de tag alphabétiquement). La classe interne TagCount et son implémentation de Comparable sont utilisées pour ce tri.
    - Les tags multiples pour un même fichier (séparés par ;) sont correctement décomposés et comptabilisés individuellement.
- **Code principal :** showPropertiesDialog(), classe interne TagCount.

4.5 Exportation de Données

L'application permet d'exporter deux types d'informations dans un fichier texte (.txt).

- **Accès :**
    - Bouton "Exporter Liste" : Exporte la liste des fichiers actuellement affichés

dans le TableView.
    - Bouton "Exporter Propriétés" : Exporte les mêmes statistiques que celles affichées dans la boîte de dialogue "Propriétés".
- **Interface :** Un FileChooser (showSaveDialog) permet à l'utilisateur de choisir l'emplacement et le nom du fichier d'exportation. L'extension .txt est ajoutée automatiquement si elle n'est pas fournie.
- **Logique (exportToFile(boolean exportProperties)) :**
    - **Si exportProperties est vrai :**
        - Récupère tous les fichiers favoris.
        - Calcule les statistiques (nombre total, auteurs uniques, tags uniques, nombre de fichiers par tag) de manière similaire à showPropertiesDialog().
        - Écrit ces informations formatées dans le fichier sélectionné.
    - **Si exportProperties est faux :**
        - Prend la liste des fichiers actuellement présents dans favoriteFilesData (ceux affichés dans le TableView, qui peuvent être le résultat d'une recherche).
        - Pour chaque fichier, écrit ses détails (Titre, Auteur, Tags, Chemin, Résumé, Commentaires) de manière formatée dans le fichier sélectionné.
    - Utilise PrintWriter et FileWriter pour l'écriture dans le fichier.
    - Une alerte informe de la réussite ou de l'échec de l'exportation.
- **Code principal :** exportToFile().

5. Structure des Classes Principales

Le code est organisé autour de la classe principale Projet_POO_AV et de quelques classes internes pour la modélisation des données et la gestion de fonctionnalités spécifiques.

5.1 Classe Projet_POO_AV

C'est la classe centrale de l'application. Elle hérite de javafx.application.Application et est donc le point d'entrée pour l'interface JavaFX.

- **main(String[] args)**: Méthode statique standard qui lance l'application JavaFX via launch(args).
- **start(Stage primaryStage)**: Méthode principale de JavaFX, appelée après l'initialisation. Elle configure la fenêtre principale (primaryStage), charge le driver JDBC, initialise la structure de la base de données (createTableIfNotExists()), met en place les composants de l'interface utilisateur (setupTableView(), createButtonPanel()), et charge les données initiales (loadFavoriteFilesAndUpdateTable()).

- **Variables d'instance clés :**
  - primaryStage: La fenêtre principale de l'application.
  - tableView: Le TableView affichant les fichiers.
  - favoriteFilesData: L'ObservableList liant les données au tableView.
- **Méthodes JDBC :** Contient toutes les méthodes pour interagir avec la base de données Oracle (e.g., getConnection(), addFavoriteFile(), getAllFavoriteFiles(), etc.). Ces méthodes gèrent l'ouverture et la fermeture des connexions, la préparation et l'exécution des requêtes SQL, et le mapping des ResultSet vers des objets FavoriteFile.
- **Méthodes UI :** Contient les méthodes pour construire et gérer l'interface utilisateur (e.g., setupTableView(), createButtonPanel(), showAddEditDialog(), showAlert()).

5.2 Classe Interne FavoriteFile

- **Attributs :** id (int), filePath (String), title (String), author (String), tags (String), summary (String), comments (String). Tous les attributs sont final car ils sont initialisés via le constructeur et ne sont pas modifiés par la suite (l'objet est immuable une fois créé, bien que les données en base puissent changer et un nouvel objet être créé pour représenter cet état).
- **Constructeur :** FavoriteFile(int id, String filePath, String title, String author, String tags, String summary, String comments). Gère les cas où author, summary, ou comments sont null en les initialisant à des chaînes vides.
- **Méthodes :** Uniquement des getters pour accéder aux valeurs des attributs.

5.3 Classe Interne SearchCriteria

- **Attributs :** title (String), author (String), tag (String). Tous les attributs sont final.
- **Constructeur :** SearchCriteria(String title, String author, String tag). Il s'assure que les valeurs nulles sont converties en chaînes vides et supprime les espaces superflus en début et fin de chaîne (trim()).

5.4 Classe Interne TagCount

Une classe utilitaire conçue pour faciliter le comptage et le tri des tags par popularité.

- **Attributs :** tag (String), count (int).
- **Constructeur :** TagCount(String tag, int count).
- **compareTo(TagCount other)**: Implémente l'interface Comparable<TagCount>. Permet de trier les objets TagCount d'abord par count (ordre décroissant) puis, en cas d'égalité de count, par tag (ordre alphabétique croissant). Ceci est utilisé dans la fonctionnalité "Propriétés" et "Exporter Propriétés".

6. Interface Utilisateur (JavaFX)

L'interface utilisateur est construite à l'aide des composants JavaFX et est conçue pour être fonctionnelle et intuitive.

6.1 Fenêtre Principale

- **Structure :** Utilise un BorderPane comme conteneur principal (mainLayout).
  - **Centre :** Le TableView (tableView) affichant la liste des fichiers favoris.
  - **Bas :** Un HBox (buttonPanel) contenant les boutons d'action principaux.
- **Titre de la fenêtre :** "Projet POO AV - Gestionnaire de Fichiers".
- **Dimensions initiales :** 800x600 pixels.

6.2 Boîtes de Dialogue

Plusieurs boîtes de dialogue modales sont utilisées pour les interactions spécifiques :

- **Ajout/Modification de Fichier (showAddEditDialog) :**
  - Utilise un GridPane pour organiser les labels et champs de saisie.
  - Comprend des TextField pour le titre, l'auteur, les tags; des TextArea pour le résumé et les commentaires; un TextField non éditable pour le chemin du fichier; et un Button ("Sélectionner...") pour ouvrir un FileChooser.
  - Des boutons "Enregistrer" et "Annuler".
- **Recherche (showSearchDialog) :**
  - Utilise un GridPane avec des TextField pour les critères de titre, auteur et tag.
  - Des boutons "Rechercher" et "Annuler".
- **Propriétés (showPropertiesDialog) :**
  - Utilise un VBox contenant un TextArea pour afficher les statistiques.
- **Confirmation de Suppression (deleteSelectedFile) :**
  - Utilise une Alert de type CONFIRMATION.
- **Notifications (showAlert) :**
  - Utilise des Alert de type INFORMATION, WARNING, ERROR pour informer l'utilisateur des résultats des opérations ou des erreurs. Notamment, pour les types ERROR et WARNING, le message est également imprimé sur la console système (System.out.println), ce qui peut être utile pour le débogage mais n'est pas standard pour une application purement GUI.
- **Sélection de Fichier (FileChooser) :**
  - Utilisé dans showAddEditDialog pour choisir un fichier à ajouter.
  - Utilisé dans exportToFile pour choisir l'emplacement de sauvegarde du fichier exporté.

7. Gestion des Erreurs et Notifications

L'application implémente plusieurs niveaux de gestion des erreurs :

- **Chargement du Driver JDBC :** Une ClassNotFoundException lors du chargement du driver Oracle est considérée comme critique. Un message est affiché sur System.err, et l'application se termine (Platform.exit()).
- **Erreurs SQL (SQLException) :**
  - **Création de Table :** L'erreur ORA-00955 (table existe déjà) est spécifiquement interceptée et gérée silencieusement (un message est loggué sur System.out). D'autres SQLException lors de la création de table affichent une alerte d'erreur.
  - **Ajout de Fichier :** L'erreur ORA-00001 (contrainte d'unicité violée pour file_path) est interceptée, et une alerte spécifique est montrée à l'utilisateur.
  - **Autres opérations JDBC (MàJ, suppression, lecture, recherche) :** Les SQLException génériques sont attrapées, un message est loggué sur System.err, et une alerte d'erreur est affichée à l'utilisateur avec le message de l'exception.
- **Erreurs d'Entrée/Sortie (IOException) :**
  - Lors de l'ouverture d'un fichier (Desktop.open()), une IOException est gérée par l'affichage d'une alerte.
  - Lors de l'exportation de fichiers, une IOException est gérée par l'affichage d'une alerte.
- **Validation des Entrées Utilisateur :**
  - Dans la boîte de dialogue d'ajout/modification, les champs obligatoires (chemin, titre, tags) sont vérifiés. Si des informations manquent, une alerte d'erreur est affichée.
- **Notifications Utilisateur (showAlert) :**
  - La méthode showAlert(Alert.AlertType type, String title, String content) centralise l'affichage des messages.
  - Pour les types INFORMATION et CONFIRMATION, une Alert JavaFX est montrée.
  - Pour les types ERROR et WARNING, en plus d'une alerte (implicitement, car le code original redirigeait vers System.out.println pour ces types, mais une Alert serait plus cohérente avec une GUI. Le code fourni *imprime sur la console* pour ERROR et WARNING au lieu d'afficher une Alert JavaFX. C'est un comportement spécifique de cette implémentation). *Correction après relecture du code fourni : Le code actuel affiche bien une Alert JavaFX pour INFORMATION et CONFIRMATION, mais pour ERROR et WARNING, il imprime un message formaté sur System.out.println. Cela signifie que les erreurs et avertissements ne sont pas présentés à l'utilisateur via des boîtes de dialogue*

graphiques, mais uniquement sur la console.