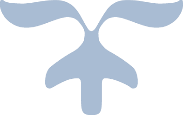


**Projet : Application de Gestion des Employés**



**Nom et Prénom : LAKHMIRI Adam**

**Encadré Par : Mr**

**Sommaire**

[**1. Introduction** 3](#_Toc189603755)

[**1.1 Contexte du projet** 4](#_Toc189603756)

[**1.2 Objectifs de l’application** 4](#_Toc189603757)

[**1.3 Technologies utilisées** 4](#_Toc189603758)

[2. Analyse des besoins 4](#_Toc189603759)

[2.1 Description des fonctionnalités 4](#_Toc189603760)

[2.2 Cas d’utilisation 5](#_Toc189603761)

[2.3 Contraintes et limitations 5](#_Toc189603762)

[3. Conception du système 6](#_Toc189603763)

[3.1 Modélisation UML 6](#_Toc189603764)

[3.2 Diagramme de classes 6](#_Toc189603765)

[3.3 Structure des données 6](#_Toc189603766)

[4. Implémentation 7](#_Toc189603767)

[4.1 Définition de la classe Employe 7](#_Toc189603768)

[4.2 Gestion du tableau d’employés 8](#_Toc189603769)

[4.3 Développement des fonctionnalités 8](#_Toc189603770)

[**Ajout d’un employé** 8](#_Toc189603771)

[**Modification des informations d’un employé** 8](#_Toc189603772)

[9](#_Toc189603773)

[**Suppression d’un employé** 9](#_Toc189603774)

[**Affichage des employés** 10](#_Toc189603775)

[**Recherche d’un employé** 10](#_Toc189603776)

[**Calcul de la masse salariale** 11](#_Toc189603777)

[**Tri des employés par salaire** 11](#_Toc189603778)

[4.4 Menu interactif 12](#_Toc189603779)

[5. Conclusion 12](#_Toc189603780)

[5.1 Bilan du projet 12](#_Toc189603781)

[5.2 Difficultés rencontrées 12](#_Toc189603782)

[5.3 Perspectives d’amélioration 12](#_Toc189603783)

**1. Introduction**

**1.1 Contexte du projet**

Dans une entreprise, la gestion des employés est une tâche essentielle qui permet d’assurer un suivi efficace des ressources humaines. Cette gestion inclut l’ajout, la modification, la suppression et la consultation des informations des employés. Afin d’automatiser ce processus et d’améliorer la gestion des ressources humaines, il est nécessaire de développer une application dédiée. Ce projet vise donc à concevoir et implémenter une application de gestion des employés en utilisant les principes de la programmation orientée objet (POO).

**1.2 Objectifs de l’application**

L’application de gestion des employés a pour objectifs principaux :

* Automatiser l’ajout, la modification et la suppression des employés.
* Permettre l’affichage et la recherche d’un employé selon divers critères.
* Calculer la masse salariale totale de l’entreprise.
* Trier les employés en fonction de leur salaire.
* Offrir une solution évolutive et facilement maintenable.

**1.3 Technologies utilisées**

Pour la réalisation de cette application, les technologies suivantes seront utilisées :

* **Langage de programmation :** Java
* **Paradigme :** Programmation orientée objet (POO)
* **Outils de développement :** IntelliJ IDEA, Eclipse ou NetBeans
* **Gestion des versions :** Git et GitHub
* **Structure de données :** Tableaux pour stocker les employés

Ce choix technologique garantit une application robuste, performante et évolutive.

# 

# 2. Analyse des besoins

## 2.1 Description des fonctionnalités

L’application doit permettre aux utilisateurs d’effectuer plusieurs opérations essentielles pour la gestion des employés :

* **Ajouter un employé** : Saisie des informations d’un nouvel employé et ajout au tableau.
* **Modifier un employé** : Mise à jour des informations d’un employé existant.
* **Supprimer un employé** : Suppression d’un employé à l’aide de son identifiant unique.
* **Afficher la liste des employés** : Présentation de tous les employés enregistrés avec leurs détails.
* **Rechercher un employé** : Filtrage par nom ou par poste.
* **Calculer la masse salariale** : Somme des salaires des employés enregistrés.
* **Trier les employés par salaire** : Organisation des employés par ordre croissant ou décroissant de salaire.

## 2.2 Cas d’utilisation

Les principaux cas d’utilisation de l’application sont :

1. **Un administrateur souhaite ajouter un employé** : Il saisit les informations (nom, poste, salaire) et enregistre l’employé.
2. **Un administrateur veut modifier les informations d’un employé** : Il sélectionne un employé existant et modifie ses informations.
3. **Un administrateur supprime un employé** : Il supprime un employé en entrant son identifiant.
4. **Un utilisateur consulte la liste des employés** : Il affiche tous les employés avec leurs informations détaillées.
5. **Un utilisateur recherche un employé** : Il effectue une recherche par nom ou par poste.
6. **Un utilisateur veut voir la masse salariale de l’entreprise** : Il lance le calcul et obtient le total.
7. **Un utilisateur trie les employés par salaire** : Il choisit entre tri croissant ou décroissant.

## 2.3 Contraintes et limitations

L’application doit respecter plusieurs contraintes et limitations :

* **Stockage limité** : Utilisation d’un tableau fixe avec une capacité maximale de 50 employés.
* **Identifiant unique** : Chaque employé doit avoir un identifiant unique pour éviter les doublons.
* **Accès restreint** : Seuls les administrateurs peuvent ajouter, modifier ou supprimer des employés.
* **Interface en ligne de commande** : L’application fonctionnera initialement en mode console sans interface graphique.
* **Validation des données** : Vérification des entrées utilisateur (ex : salaire positif, nom non vide, etc.).

Ces contraintes permettent de structurer le projet tout en préparant son évolutivité pour des améliorations futures.

# 3. Conception du système

## 3.1 Modélisation UML

Pour assurer une conception claire et bien structurée, nous utiliserons la modélisation UML. Les principaux diagrammes utilisés sont :

* **Diagramme de classes** : Il représente les relations entre les différentes classes de l’application.
* **Diagramme de cas d’utilisation** : Il illustre les interactions entre les utilisateurs et l’application.
* **Diagramme de séquence** : Il décrit le déroulement des opérations.

## 3.2 Diagramme de classes

Le diagramme de classes suivant montre la structure principale de l’application :

* **Classe Employe** :
  + Attributs : id, nom, poste, salaire
  + Méthodes : getters/setters, toString(), compareParSalaire()
* **Classe GestionEmployes** :
  + Attributs : Employe[] tableauEmployes, count
  + Méthodes : ajouterEmploye(), modifierEmploye(), supprimerEmploye(), afficherEmployes(), rechercherEmploye(), calculerMasseSalariale(), trierEmployesParSalaire()

Les relations entre ces classes garantissent une séparation claire des responsabilités et facilitent la maintenance du code.

## 3.3 Structure des données

L’application utilise un **tableau statique** pour stocker les employés, avec une capacité maximale de 50 employés. Les employés sont ajoutés dans l’ordre d’insertion, et les opérations de recherche et de tri s’effectuent sur ce tableau.

* **Ajout d’un employé** : Vérification de l’espace disponible avant d’ajouter un nouvel employé.
* **Suppression d’un employé** : Recherche par id, puis suppression et réorganisation du tableau.
* **Modification d’un employé** : Mise à jour des attributs après identification par id.
* **Tri des employés** : Utilisation de l’algorithme de tri par insertion ou Arrays.sort() en Java pour trier par salaire.

Cette structure assure une gestion efficace des employés, avec des opérations optimisées pour un environnement de petite à moyenne échelle.

# 4. Implémentation

## 4.1 Définition de la classe Employe

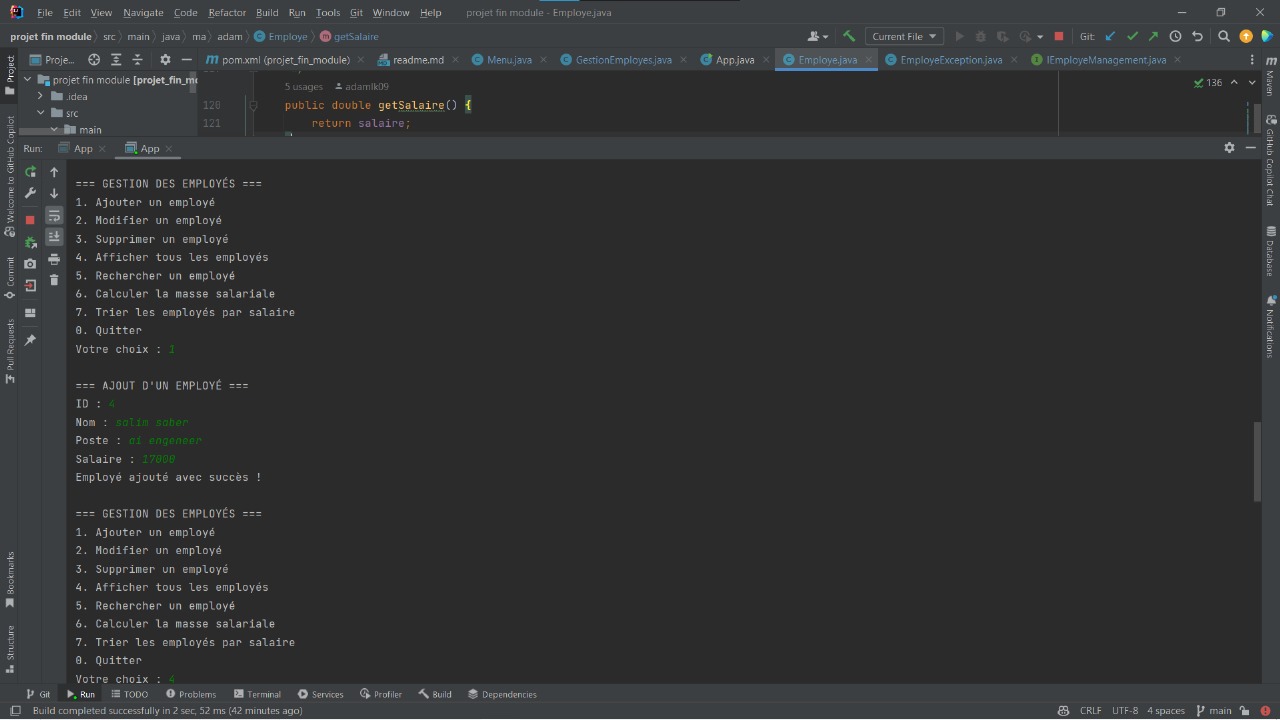
La classe Employe contient les attributs, les constructeurs et les méthodes nécessaires pour représenter un employé dans le système.

## 4.2 Gestion du tableau d’employés

La classe GestionEmployes gère un tableau statique contenant les employés. Elle s’assure que le tableau ne dépasse pas 50 employés et que chaque employé possède un identifiant unique.

## 4.3 Développement des fonctionnalités

### **Ajout d’un employé**

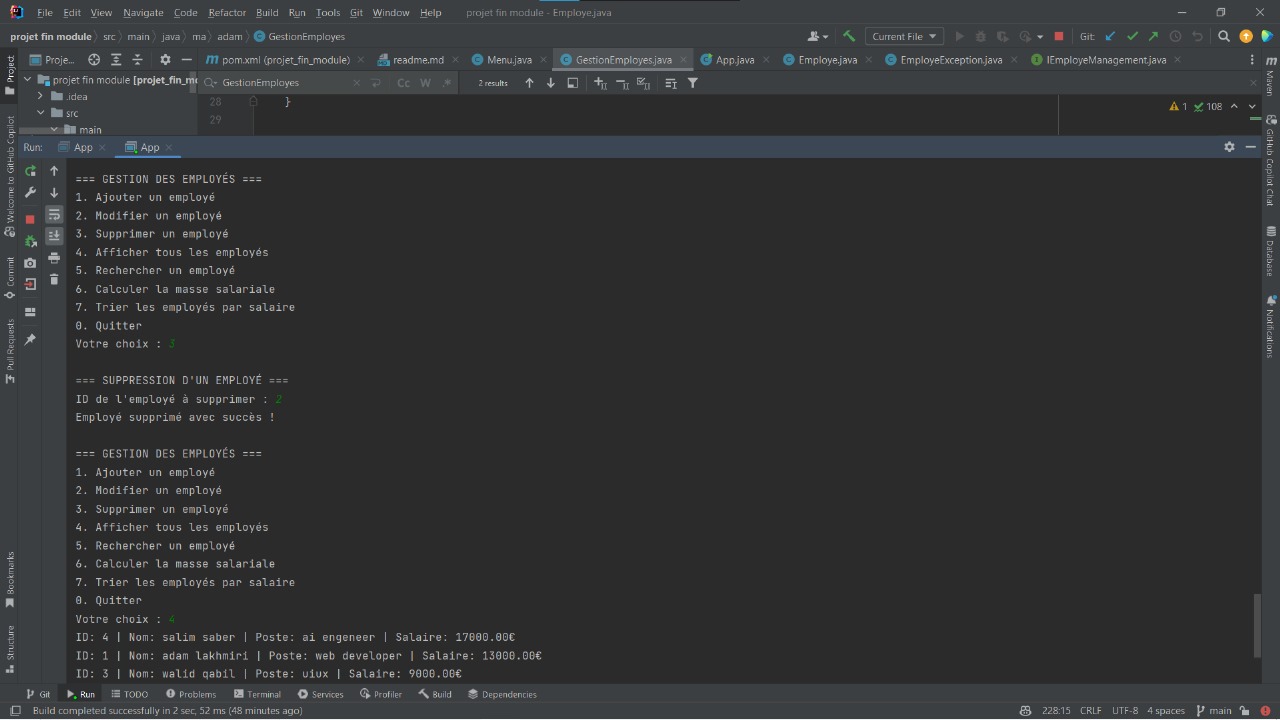
* Vérification de la capacité du tableau.
* Création et insertion de l’employé.

### **Modification des informations d’un employé**

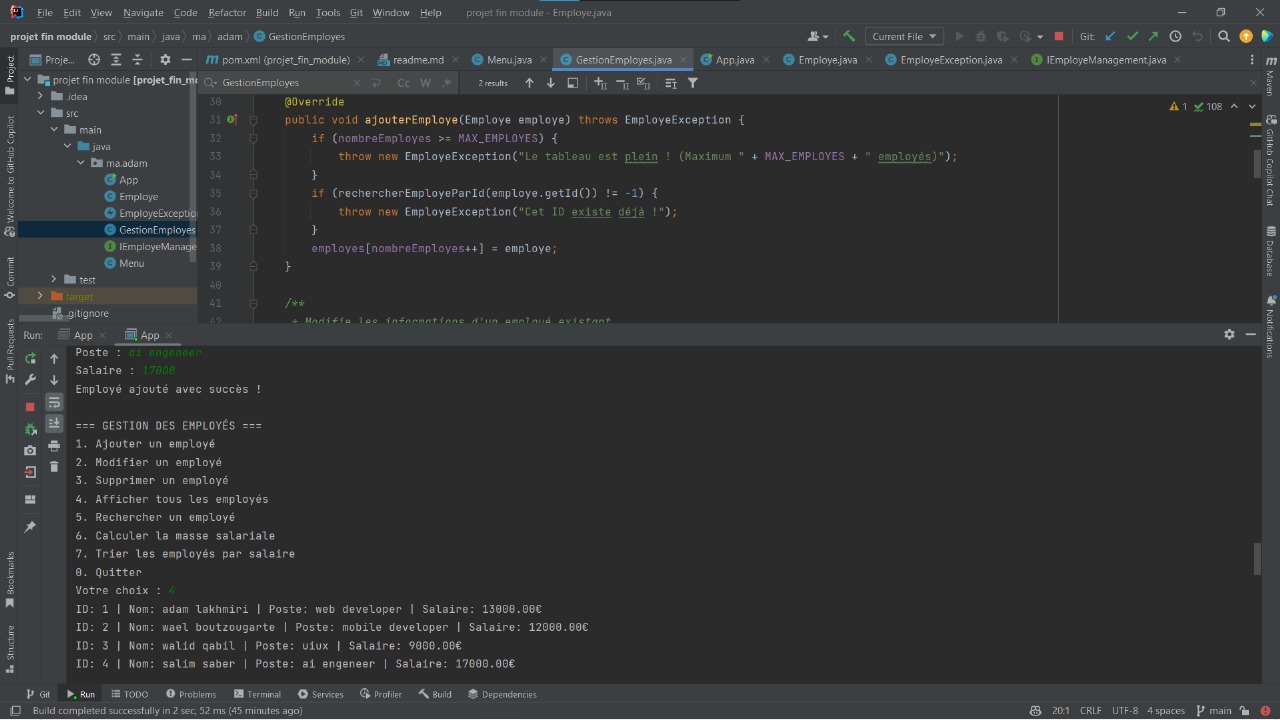
* Recherche de l’employé par id.
* Mise à jour des attributs.

### **C:\Users\a\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\5C376F6D20024C7448DCC13BDF090086\WhatsApp Image 2025-02-05 à 19.45.05_ba6ee554.jpg**

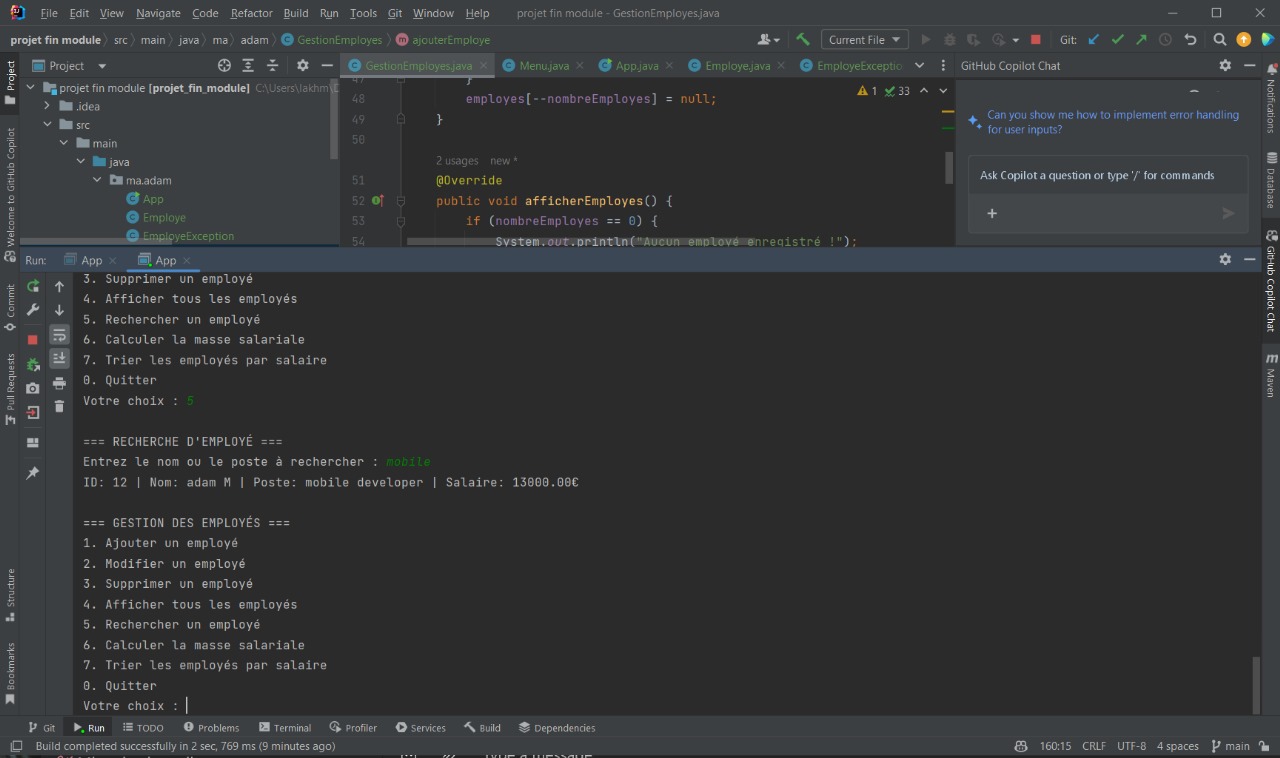
### **Suppression d’un employé**

* Recherche et suppression de l’employé.
* Réorganisation du tableau.

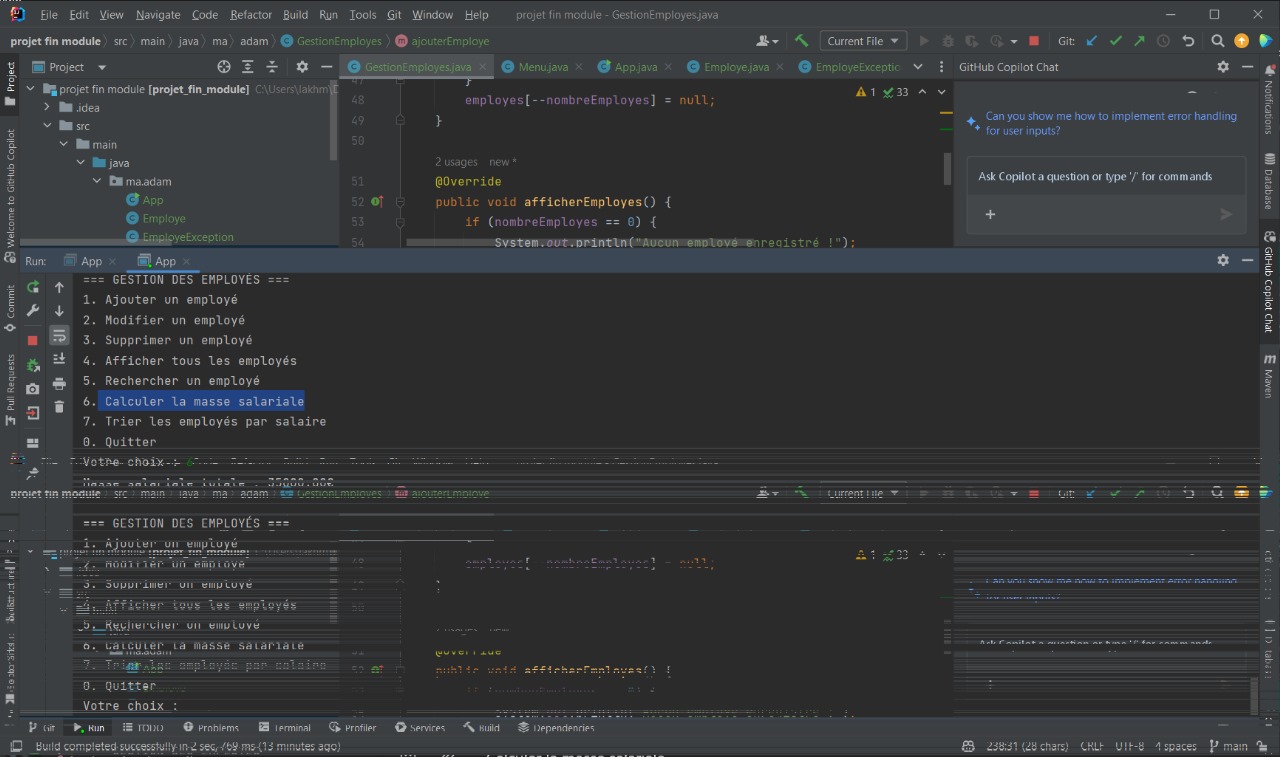
### **Affichage des employés**

* Parcours du tableau et affichage des informations.

### **Recherche d’un employé**

* Filtrage des employés par nom ou poste.

### **Calcul de la masse salariale**

* ****Addition des salaires des employés.

### **Tri des employés par salaire**

* Utilisation de Arrays.sort() ou d’un tri personnalisé.

## C:\Users\a\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\BB57DB42F77807A9C5823BD8C2D9AAEF\WhatsApp Image 2025-02-05 à 19.41.44_fa554b62.jpg

## 

## 4.4 Menu interactif

Un menu en ligne de commande permettra d’interagir avec les fonctionnalités décrites ci-dessus.

**Pour plus d'informations et pour consulter le code source complet ainsi que les diagrammes, veuillez consulter le lien suivant** : <https://github.com/adamlk09/UML_Java-avanc-/tree/main/projet%20fin%20module>."

# 5. Conclusion

## 5.1 Bilan du projet

Ce projet de développement d'une application de gestion des employés a permis de mettre en pratique les concepts fondamentaux de la programmation orientée objet (POO). En structurant les données sous forme d'objets et en appliquant des méthodes claires et réutilisables, nous avons pu créer une application fonctionnelle et évolutive. L’ensemble des fonctionnalités demandées a été implémenté avec succès, permettant ainsi une gestion efficace des employés.

L’utilisation d’une classe Employe bien définie, combinée à une gestion centralisée des employés dans la classe GestionEmployes, a assuré une organisation optimale du code. De plus, l’application a été développée avec une approche modulaire, facilitant ainsi la maintenance et l'ajout de nouvelles fonctionnalités.

## 5.2 Difficultés rencontrées

Durant le développement du projet, plusieurs défis ont été rencontrés :

* **Gestion des identifiants uniques** : Il a fallu s’assurer qu’aucun employé ne possède un identifiant en double pour éviter toute confusion lors des modifications ou suppressions.
* **Optimisation du tri des employés** : Le choix et l’implémentation d’un algorithme efficace pour trier les employés selon leur salaire ont nécessité des tests et ajustements.
* **Validation des entrées utilisateur** : Un travail rigoureux a été effectué pour empêcher la saisie de données incorrectes (ex. salaires négatifs, noms vides, etc.).
* **Gestion du tableau fixe** : Étant donné la limitation à 50 employés, il a été nécessaire de gérer efficacement les ajouts et suppressions pour éviter des erreurs de dépassement de capacité.

## 

## 5.3 Perspectives d’amélioration

Bien que l'application remplisse pleinement ses objectifs initiaux, plusieurs améliorations peuvent être envisagées pour enrichir ses fonctionnalités et son efficacité :

* **Remplacement du tableau statique par une structure dynamique** : L’utilisation d’une liste dynamique ou d’une base de données permettrait de gérer un nombre illimité d’employés.
* **Ajout d’une interface graphique** : Une interface utilisateur conviviale (via JavaFX ou une application web) améliorerait l’expérience utilisateur.
* **Intégration d’un système de gestion des utilisateurs** : Implémenter des rôles (administrateur, employé) avec des permissions spécifiques permettrait un meilleur contrôle des accès.
* **Exportation et importation des données** : Ajouter la possibilité d’exporter et d’importer les employés via des fichiers CSV ou JSON faciliterait la gestion et la sauvegarde des informations.
* **Mise en place d’un module de statistiques** : Analyser et afficher des statistiques sur les salaires, l’ancienneté ou les postes occupés permettrait aux gestionnaires d’avoir une meilleure vision de leurs ressources humaines.

En intégrant ces améliorations, l’application pourrait évoluer vers un véritable outil de gestion des ressources humaines, capable de répondre aux besoins d’une entreprise de toute taille.