Rapport TP controle:

Cahier de charge: **Mini-Projet Java – Gestion Simplifiée des Stages**  
**--------------------------------------------------------------**  
:cercle_blanc: **Contexte :**  
Une entreprise souhaite développer une application Java pour gérer ses stagiaires et leurs stages. L'objectif est de permettre une gestion simple et efficace des informations relatives aux stagiaires et aux stages, en utilisant les concepts fondamentaux de programmation orientée objet.:cercle_blanc: **Objectifs du projet :**

1. Concevoir un système modulaire basé sur une architecture orientée objet.
2. Manipuler des collections pour stocker et gérer les données des stagiaires et des stages.
3. Implémenter des interfaces pour définir des comportements communs.
4. Exploiter l'héritage et le polymorphisme pour structurer les classes.

:cercle_blanc: **Fonctionnalités demandées :**  
**1. Gestion des Stagiaires :**

Ajouter un stagiaire en fournissant les informations suivantes :

Nom

Prénom

Niveau d'étude (exemple : Licence, Master, etc.).

Supprimer un stagiaire par son nom.

Afficher la liste de tous les stagiaires avec leurs informations.

**2. Gestion des Stages :**

Ajouter un stage avec les informations suivantes :

Titre du stage.

Durée (en jours).

Associer un ou plusieurs stagiaires à un stage.

Afficher la liste des stages, en précisant les stagiaires associés à chaque stage.

**3. Recherche et Statistiques :**

Rechercher un stagiaire par son nom ou son niveau d'étude.

Calculer le nombre total de stages.

Afficher le nombre de stagiaires associés à chaque stage.

:cercle_blanc: **Contraintes Techniques :**  
**1. Héritage :**

Une classe **Personne** doit être utilisée comme classe mère. Elle contiendra les attributs et méthodes communs pour les personnes.

Une classe **Stagiaire** doit hériter de la classe **Personne**.

**2. Polymorphisme :**

Une méthode abstraite dans la classe **Personne** doit être implémentée différemment dans les sous-classes.

Les informations des stagiaires doivent être affichées de manière polymorphique.

**3. Interfaces :**

Une interface **Gestion** doit être créée pour définir les méthodes suivantes :

ajouter() : Ajouter un élément (stagiaire ou stage).

supprimer() : Supprimer un élément.

afficher() : Afficher les informations d'un élément ou d'une collection.

**4. Collections :**

Les stagiaires et les stages doivent être stockés dans des collections dynamiques (comme **ArrayList**).

:cercle_blanc: **Exigences supplémentaires :**

1. L'application doit utiliser un menu simple en mode console pour naviguer entre les options :

**1. Ajouter un stagiaire**

**2. Supprimer un stagiaire**

**3. Afficher tous les stagiaires**

**4. Ajouter un stage**

**5. Associer un stagiaire à un stage**

**6. Afficher tous les stages avec les stagiaires associés**

**7. Rechercher un stagiaire**

**8. Quitter l'application**

1. Les données doivent être sauvegardées en mémoire uniquement (pas de base de données ou de fichiers nécessaires).
2. Validez les entrées utilisateur pour éviter les doublons ou les erreurs.

### **1. Contexte du Projet**

Une entreprise souhaite développer une application Java permettant de gérer efficacement les stagiaires et leurs stages. L'objectif est de concevoir une application simple en mettant en œuvre les concepts fondamentaux de la programmation orientée objet (POO).

### **2. Objectifs du Projet**

* Créer un système modulaire basé sur les principes de la POO.
* Manipuler des collections dynamiques pour stocker et gérer les données des stagiaires et des stages.
* Utiliser les interfaces pour définir des comportements communs.
* Exploiter les mécanismes d'héritage et de polymorphisme pour structurer les classes.

### **3. Fonctionnalités Implémentées**

#### **3.1. Gestion des Stagiaires**

* **Ajout d'un stagiaire** : Les informations nécessaires incluent le nom, le prénom et le niveau d'étude (Licence, Master, etc.).
* **Suppression d'un stagiaire** : Suppression possible en fonction du nom du stagiaire.
* **Affichage des stagiaires** : Une liste affiche les stagiaires avec leurs informations détaillées.

#### **3.2. Gestion des Stages**

* **Ajout d'un stage** : Informations nécessaires : titre du stage et durée en jours.
* **Association des stagiaires à un stage** : Possibilité d'associer un ou plusieurs stagiaires à un stage.
* **Affichage des stages** : Liste des stages avec les stagiaires associés.

#### **3.3. Recherche et Statistiques**

* **Recherche d'un stagiaire** : Recherche possible par nom ou par niveau d'étude.
* **Calcul du nombre total de stages.**
* **Statistiques sur les stagiaires** : Affichage du nombre de stagiaires associés à chaque stage.

### **4. Contraintes Techniques Respectées**

#### **4.1. Héritage**

* Une classe **Personne** a été créée comme classe parente pour contenir les attributs et méthodes communs (par exemple, nom et prenom).
* Une classe **Stagiaire** hérite de la classe **Personne**, ajoutant des attributs spécifiques tels que niveauDetude.

#### **4.2. Polymorphisme**

* Une méthode abstraite, afficherInformations(), définie dans la classe **Personne**, a été implémentée de manière différente dans les sous-classes.
* L'affichage des informations utilise le polymorphisme pour adapter le comportement en fonction du type d'objet.

#### **4.3. Interfaces**

* Une interface **Gestion** définit les méthodes communes :
  + ajouter() : pour ajouter des éléments (stagiaires ou stages).
  + supprimer() : pour supprimer des éléments.
  + afficher() : pour afficher des informations.

#### **4.4. Collections Dynamiques**

* Les stagiaires et les stages sont stockés dans des collections de type **ArrayList** pour permettre une gestion flexible et dynamique des données.

### **5. Interface Utilisateur**

L'application utilise un menu simple en mode console pour naviguer entre les différentes options :

1. Ajouter un stagiaire
2. Supprimer un stagiaire
3. Afficher tous les stagiaires
4. Ajouter un stage
5. Associer un stagiaire à un stage
6. Afficher tous les stages avec leurs stagiaires
7. Rechercher un stagiaire
8. Quitter l'application

#### **Validation des Entrées :**

* Les doublons sont évités en vérifiant que les noms des stagiaires ou des stages ne se répètent pas.
* Les entrées utilisateur sont validées pour minimiser les erreurs.

### **6. Structure du Code**

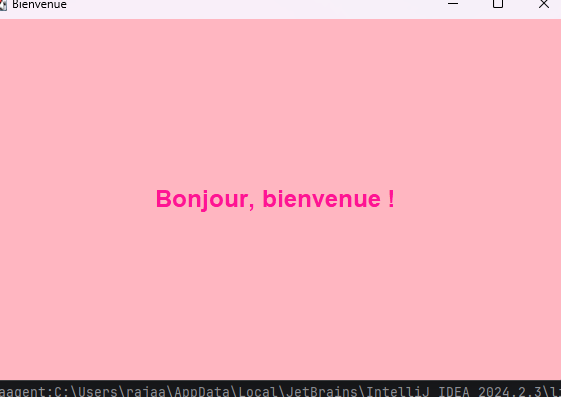
#### **6.1. Diagramme de Classes**

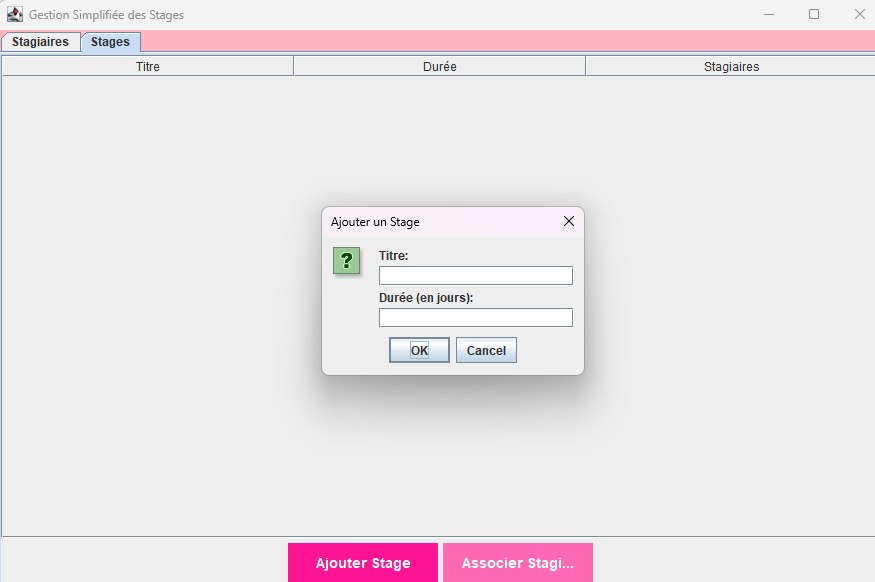
* **Personne** (classe mère) :
  + Attributs : nom, prenom
  + Méthodes : afficherInformations() (abstraite)
* **Stagiaire** (sous-classe) :
  + Attributs : niveauDetude
  + Méthodes : implémentation de afficherInformations()
* **Stage** (classe indépendante) :
  + Attributs : titre, duree, stagiairesAssocies (de type ArrayList)
  + Méthodes : gestion des associations, affichage des informations
* **Gestion (Interface)** :
  + Méthodes : ajouter(), supprimer(), afficher()

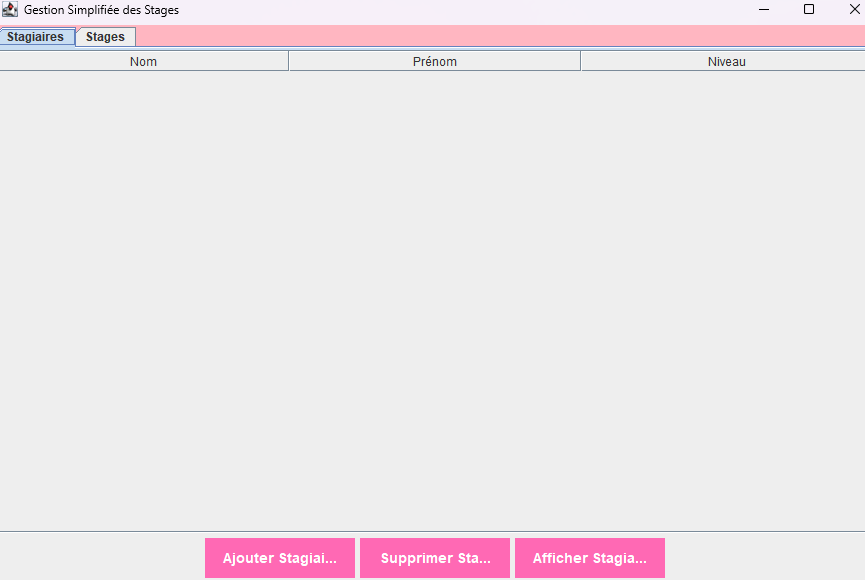
#### **6.2. Organisation des Packages**

* **main** : Contient la classe principale avec la méthode main.
* **model** : Contient les classes Personne, Stagiaire et Stage.
* **service** : Contient les interfaces et les méthodes de gestion.

Extrait de l’éxecution:







### **7. Limitations et Améliorations Possibles**

* **Limitations** :
  + Interface utilisateur limitée au mode console.
* **Améliorations Futures** :
  + Intégration d'une base de données pour la persistance des données.
  + Création d'une interface graphique pour améliorer l'expérience utilisateur.
  + Ajout de rapports détaillés exportables (par exemple, en PDF).

### **8. Conclusion**

Ce projet a permis de développer une application modulaire et fonctionnelle en Java, en mettant en pratique les concepts fondamentaux de la POO. Bien que l'application soit limitée à une utilisation en mémoire et en mode console, elle constitue une base solide pour des extensions futures.

Fait par:Rajae Fdili