Révision finale

On souhaite réaliser une application de gestion des employés et des tâches.
On considère que :

- Une tâche est caractérisée par : un numéro, un intitulé, une description, une priorité (entier), date de début, date de fin, durée, état (sa valeur peut être : en cours, suspendu, terminée, annulé, programmée).
- Un employé est caractérisé par : un identifiant, un nom, un prénom, un CIN, une date de naissance, une spécialité, un salaire, une liste des tâches.

1. **Déclarer une enum** Etat dont les valeurs sont en cours, suspendu, terminée, annulé, programmée (en majuscule)
2. **Déclarer une classe Tâche qui possède**
   a. Des attributs privés (voir la description)
   b. Un constructeur avec paramètre
   c. La méthode toString
   d. La méthode equals : deux tâches sont égales si elles possèdent le même numéro et le même intitulé.
   e. Guetteurs et setters. Le setter de la priorité de tâche ne peut accepter qu'une valeur entre 1 et 5.
3. **Déclarer une classe abstrait employé qui implémente comparable et qui possède :**
   a. Des attributs privés.
   b. Un constructeur avec paramètre
   c. La méthode toString
   d. **public boolean ajouterTache(Tache t)** la tâche t ne peut pas être ajouté si il existe un autre tâche dans la liste qui porte le même identifiant. La méthode retourne true si l'ajout est effectué, false sinon.
   e. **public boolean supprimerTache(int numero**) permet de supprimer la tâche qui possède le numéro passé en paramètre et retourne true. S'il n'existe aucune tâche ayant le numéro passé en paramètre, la méthode retournera false.
   f. **Méthodes abstraites :**
      i. **public abstract double calculerPrime(); :** Calcul de la prime selon la classe dérivée.
      ii. **public abstract void afficherDetails();** : Affiche des informations spécifiques à la classe dérivée.
      iii. **public abstract double calculerCharge();** : Retourne la charge de travail ou la performance.
   g. Des méthodes qui utilisent les streams :
      i. **public void afficherParSpecialite()** : permet d'afficher les employés par spécialité.
      ii. **public int nombreTache() :** retourne le nombre des employés qui ont un nombre de tâche avec l'état terminé supérieur à 10
      iii. **public int nombreEmploye() :** retourne le nombre des employés qui ont la spécialité « informatique »
      iv. **public Map<Etat, List<Tache>> getTache(Liste<Tache> l) :** permet de retourner un map des tâches groupée par état.

4. Créer les classes dérivées suivantes :

a. Classe Technicien : le Technicien représente un employé spécialisé dans des tâches techniques. Ses responsabilités peuvent être liées à la maintenance, au support ou à l'exécution de tâches techniques spécifiques.

- Attributs spécifiques :
  - **int niveauTechnique** : Niveau technique (par exemple : 1 pour débutant, 5 pour expert).
  - **int experience** : Années d'expérience dans le domaine technique.
  - **List<String> certifications :** Liste des certifications techniques obtenues (ex. : Cisco, AWS).
- Méthodes spécifiques :
  - **public void ajouterCertification(String certification)** : Ajoute une nouvelle certification à la liste.
  - **public void afficherCertifications() :** Affiche les certifications avec leur niveau d'expertise.
  - **public double calculerPrime() :** La prime pourrait dépendre du niveau technique et du nombre d'années d'expérience. Par exemple : (niveauTechnique * 200) + (experience * 100);
  - **public int getNombreTachesTechniques()** : Retourne le nombre de tâches où la priorité est élevée (supérieure à 3).

b. Classe Ingénieur : L'Ingénieur est un employé qui gère ou exécute des projets. Ses tâches sont souvent liées à la conception, la planification ou la réalisation de projets.

- Attributs spécifiques :
  - **List<String> projetsRealises** : Liste des projets réalisés par l'ingénieur.
  - **String specialisation :** Spécialisation technique ou fonctionnelle (ex. : IA, réseaux, génie logiciel).
  - **int niveauResponsabilite** : Niveau de responsabilité dans l'organisation (1 : débutant, 5 : senior/chef de projet).
- Méthodes spécifiques :
  - **public void ajouterProjet(String projet) : Ajoute** un projet réalisé à la liste.
  - **public double calculerPerformance()** : Calcule une note de performance basée sur le nombre de projets terminés et le niveau de responsabilité.
    Exemple :projetsRealises.size() * 100 + (niveauResponsabilite * 50);
  - **public void afficherDetailsProjet() :** Affiche les projets réalisés et leur spécialisation associée.
  - **public double calculerPrime()** : La prime pourrait dépendre du nombre de projets réalisés et du niveau de responsabilité.
    Exemple : (projetsRealises.size() * 300) + (niveauResponsabilite * 500);

c. Classe Administrateur : L'Administrateur est responsable de la gestion interne des équipes ou de l'organisation (RH, finance, logistique, etc.).

- Attributs spécifiques :
  - i**nt nombreEquipes** : Nombre d'équipes gérées.
  - **int dureeService :** Nombre d'années de service dans l'entreprise.
  - **double budgetAnnuel :** Budget annuel géré par l'administrateur.

- Méthodes spécifiques :
  - **public double calculerBudgetParEquipe() :** Retourne le budget moyen par équipe. budgetAnnuel / nombreEquipes;
  - **public double calculerPrime()** : La prime dépend du nombre d'équipes et de la durée de service. Exemple : (nombreEquipes * 1000) + (dureeService * 200);
  - **public void afficherRapportGestion() :** Affiche un rapport sur les équipes gérées, leur budget et les performances.
  - **public void augmenterBudget(double pourcentage) :** Augmente le budget annuel de l'administrateur d'un certain pourcentage.

d. Classe Commercial : Le Commercial est responsable de la gestion des ventes, du chiffre d'affaires, et des relations avec les clients.

- Attributs spécifiques :
  - **double chiffreAffaires : Le** chiffre d'affaires généré par le commercial.
  - **int nombreClients :** Nombre de clients gérés.
  - **double tauxCommission :** Taux de commission (%).
- Méthodes spécifiques :
  - **public double calculerCommission() :** Calcule la commission du commercial basée sur le chiffre d'affaires. Exemple : chiffreAffaires * (tauxCommission / 100);
  - **public void ajouterClient() :** Incrémente le nombre de clients gérés.
  - **public double calculerPrime() :** La prime pourrait être basée sur le chiffre d'affaires et le nombre de clients. Exemple : (chiffreAffaires * 0.05) + (nombreClients * 100);
  - **public void afficherStatistiquesVentes() :** Affiche les statistiques de ventes (CA, commission, nombre de clients).

5. **Classe GestionEmployes** Cette classe permet de manipuler une liste d'employés. Elle possède une liste des employés et les méthodes suivantes basées sur les streams :
   a. **public List<Employe> trierParSalaire() :** Retourne la liste des employés triée par salaire décroissant.
   b. **public Map<String, List<Employe>> regrouperParSpecialite() :** Regroupe les employés par spécialité.
   c. **public long nombreEmployesParSpecialite(String specialite)** : Retourne le nombre d'employés d'une spécialité donnée.
   d. **public Optional<Employe> meilleurEmploye() : Retourne** l'employé ayant la prime la plus élevée.

6. **Classe Main :** créer une classe main pour tester votre code.
7. Récrire la classe GestionEmployé en considérant un set des employés au lieu d'une liste.
8. Refaire la même chose mais cette fois ci on considère un Map dont la valeur c'est l'identifiant est un entier (identifiant) et la valeur est de type Employé.