1. Classe Entreprise

1.1. Déclarer au niveau de la classe Entreprise les constantes de classe suivantes :
   a) SALAIRE_BASE : Double: 1480.27
   b) NB_CONGES_BASE : Integer: 25
   c) INDICE_MANAGER : Double: 1.3
   d) PRIME_MANAGER_PAR_TECHNICIEN : Double: 250.0
   e) PRIME_ANCIENNETE : Double: 100.0

1.2. Déclarer au niveau de la classe Entreprise, une méthode primeAnnuelleBase, publique et statique calculant la prime de base pour tous les employés de l'entreprise de la manière suivante (Utiliser la classe LocalDate pour obtenir l'année courante) :
Prime = 50% de l'année en cours. Ex : 2017 : 2017 / 2 = 1008.5

2. Classe Employe

2.1. Rendre la classe Employe abstraite

2.2. Déclarer dans la classe Employe une méthode abstraite getPrimeAnnuelle retournant un Double

2.3. Ajouter dans la classe Employe les champs suivants avec leur getters/setters
   a) nom : String
   b) prenom : String
   c) matricule : String
   d) dateEmbauche : LocalDate
   e) salaire : Double

2.4. Définir un constructeur par défaut dans la classe Employé, puis un constructeur avec l'ensemble des arguments précédemment créés, dans le même ordre.

2.5. Déclarer et développer la méthode getNombreAnneeAnciennete calculant le nombre d'année d'ancienneté d'un employé. Faire en sorte qu'elle ne puisse être redéfinie dans d'éventuelles sous-classes. Note : Un employé embauché en 2017 a une ancienneté de 0.

2.6. Modifier le setter de dateEmbauche pour lever une Exception avec le message "La date d'embauche ne peut être postérieure à la date courante" lorsque la date d'embauche est postérieure à la date courante.

2.7. Développer une méthode getNbConges retournant la constante de classe NB_CONGES_BASE de la classe Entreprise.

2.8. Redéfinir la méthode toString (héritée d'Object) pour afficher un employé de la manière suivante : "Employe{nom='nom', prenom='prenom', matricule='12345', dateEmbauche=1970-01-01, salaire=500.0}"

2.9. Redéfinir la méthode equals (héritée d'Object) testant l'égalité sur l'ensemble des attributs de la classe Employe

2.10. Redéfinir la méthode hashCode (héritée d'Object) en utilisant Objects.hash(...) et en respectant l'ordre nom, prenom, matricule, dateEmbauche, salaire.

2.11. Coder la méthode augmenterSalaire prenant en paramètre un pourcentage d'augmentation de type Double et augmentant l'attribut salaire du pourcentage passé en paramètre. Ex : un salaire de 500.0, avec une augmentation de 0.50, cela donne un salaire de 750.0

3. Classe Technicien

3.1. Comme pour la classe Commercial, faire hériter la classe Technicien et implémenter la méthode abstraite getPrimeAnnuelle pour qu'elle retourne quelque chose et que la compilation passe (la méthode sera implémentée plus tard). Ajouter un constructeur par défaut.

3.2. Modifier la classe Technicien pour ajouter un attribut grade de type int avec son getter et son setter.

3.3. Créer un constructeur pour la classe Technicien qui initialise tous les attributs hérités de la classe Employe en faisant appel au constructeur d'Employe et qui initialise également l'attribut grade.

3.4. Modifier le setter de l'attribut grade pour qu'il lève une exception de la classe TechnicienException (à créer) et dont le message est : "Le grade doit être compris entre 1 et 5 : X, technicien : Technicien{grade=X} Employe{nom='NOM', prenom='PRENOM', matricule='MATRICULE', dateEmbauche=DATE, salaire=SALAIRE}"
Avec X = valeur incorrecte passée au setter et NOM, PRENOM... les valeurs des attributs d'Employe.
Astuce : Ajouter une méthode toString à Technicien

3.5. Redéfinir le setter de l'attribut salaire pour qu'il renvoie la valeur de l'attribut salaire, auquel on ajoute la bonification du grade qui est égale à une augmentation de X0% par rapport au salaire de base :
Ex : Grade 3 : 30% d'augmentation : 1000.0 de salaire avec grade 1 : 1100.0.

3.6. Redéfinir le getter de nbConges pour retourner le nombre de congés de base + autant de congés que d'année d'ancienneté.

3.7. Modifier le code de la méthode getPrimeAnnuelle pour qu'elle renvoie la prime annuelle de base à laquelle on ajoute un pourcentage en fonction du grade (idem exo 3.5) ainsi que la prime d'ancienneté multipliée par le nombre d'année d'ancienneté.

3.8. Implémenter l'interface Comparable pour que l'on puisse comparer deux Techniciens en fonction de leur grade plus le grade est haut, plus le technicien est compétent.

4. Classe Commercial

4.1. Faire en sorte que la classe Commercial hérite de la classe Employé. Analyser le message d'erreur remonté par l'IDE et utiliser l'IDE pour résoudre le problème.

4.2. Modifier la classe Commercial pour ajouter un attribut caAnnuel de type Double avec son getter et son setter.

4.3. Modifier la méthode getPrimeAnnuelle précédemment générée par l'IDE pour que la prime soit égale à 5% du caAnnuel, avec un minimum de 500 € même en cas de chiffre d'affaire nul. Faire en sorte que la prime soit toujours arrondi à l'euro supérieur (Voir la classe Math)

4.4. Créer un constructeur pour la classe Commercial qui initialise tous les attributs hérités de la classe Employe en faisant appel au constructeur d'Employe et qui initialise également l'attribut caAnnuel.

4.5. Surcharger la méthode equals pour permettre de tester l'égalité entre deux instances de la classe Commercial. Appeler la méthode equals de la classe Employe.

4.6. Ajouter un attribut Integer performance. Ajouter une méthode performanceEgale prenant un Integer en paramètre qui renvoie true si la performance du commercial est égale à celle passée en paramètre, false sinon.

4.7. Créer un enum note avec les valeurs INSUFFISANT, PASSABLE, BIEN, TRES_BIEN et créer une méthode equivalenceNote (sans utiliser de if) traduisant une performance en Note :

a) Si performance = 0 ou 50 : INSUFFISANT
b) Si performance = 100 : PASSABLE
c) Si performance = 150 : BIEN
d) Si performance = 200 : TRES_BIEN

5. Classe Manager

5.1. Comme pour la classe Commercial, faire hériter la classe Manager d'Employe et implémenter la méthode abstraite getPrimeAnnuelle pour qu'elle retourne quelque chose et que la compilation passe (la méthode sera implémentée plus tard)

5.2. Modifier la classe Manager pour ajouter un attribut equipe permettant de stocker un ensemble non ordonné de techniciensb avec son getter et son setter

5.3. Ajouter une méthode ajoutTechnicienEquipe qui prend en paramètre un technicien et qui l'ajoute dans l'équipe

5.4. Surcharger le setter de l'attribut salaire pour qu'il renvoie la valeur du salaire multipliée par l'index manager, auquel on ajoute 10% (sur le salaire passé en paramètre) par membre d'équipe

5.5. Modifier le code de la méthode getPrimeAnnuelle pour qu'elle renvoie la prime de base, à laquelle on ajoute la prime du manager en fonction du nombre de membres de son équipe (en utilisant Entreprise.PRIME_MANAGER_PAR_TECHNICIEN).

5.6. Surcharger la méthode augmenterSalaire pour systématiquement augmenter le salaire de l'équipe d'un manager avant d'augmenter le salaire du manager…

5.7. Surcharger ajoutTechnicienEquipe pour permettre l'ajout d'un technicien en passant directement les paramètres nom, prenom, matricule, date, salaire et grade

5.8. Ajouter une méthode equipeParGrade renvoyant une liste des techniciens de l'équipe triée par grade décroissant en utilisant les Streams et les lambdas

5.9. Ajouter une méthode salaireEquipeGrade1 qui renvoie la somme des salaires des membres de l'équipe dont le grade est égal à 1 en une ligne avec des lambdas

6. Bonus

6.1. Créer la classe générique Unite ayant deux attributs :

a) responsable de type générique et
b) membres, qui est un hashset de type générique.

Générer les getters/setters
Créer un constructeur par défaut, un constructeur avec un membre qui affecte ce membre en tant que responsable et l'ajoute aux membres
Ajouter une méthode ajouterMembre qui peut prendre autant de type générique que l'on veut
Générer une méthode toString