Nom et prénom

Contrôle Java avancé

**Travail à faire :** Compléter les vides, respecter les commentaires.

**public** **class** Employe **implements** Cloneable {

**private** String nom;

**private** String prenom;

**private** **int** matricule;

**private** **double** salaire;

**private** Departement departement;

// Déclarer l’énumération **Departement** contenant les valeurs :

// ACHAT, COMMERCIAL, INFORMATIQUE, RH;

// associer à chaque valeur une primeAnnuel : ACHAT-> 2000,

// COMMERCIAL->3000, INFORMATIQUE->5000, RH(2000)

**public** **static** **enum** Departement {

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………………… }

**public** String getNom() { **return** nom; }

**public** **void** setNom(String nom) { **this**.nom = nom; }

**public** String getPrenom() { **return** prenom; }

**public** **void** setPrenom(String prenom) {

**this**.prenom = prenom; }

**public** **int** getMatricule() { **return** matricule; }

**public** **void** setMatricule(**int** matricule) {

**this**.matricule = matricule; }

**public** **double** getSalaire() { **return** salaire; }

**public** **void** setSalaire(**double** salaire) {

**this**.salaire = salaire ; }

**public** Departement getDepartement() {

**return** departement; }

**public** **void** setDepartement(Departement departement) {

**this**.departement = departement;

}

**public** Employe(String nom, String prenom, **int** matricule, **double** salaire, Departement departement) {

**super**();

**this**.nom = nom;

**this**.prenom = prenom;

**this**.matricule = matricule;

**this**.salaire = salaire;

**this**.departement = departement;

}

@Override

**public** String toString() {

**return** "Employe [nom=" + nom + ", prenom=" + prenom + ", matricule=" + matricule + ", salaire=" + salaire

+ ", departement=" + departement + "]";

}

// redéfinir la méthode equals permettant de tester l’égalité // de deux employés.

@Override

**public** **boolean** equals(Object obj) {

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

}

// redéfinir la méthode clone permettant de dupliquer

// un employé.

@Override

**public** Employe clone() {

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

}

// Déclarer la méthode afficherPrimeDepartement permettant

// d’afficher les départements et leur primeAnnel ;

**public** **void** afficherPrimeDepartement() {

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

}

}

**//------------------------------------------------//**

Soit la classe CollectionEmploye:

**public** **class** CollectionEmploye {

**private** **static** List<Employe> *listeEmploye*=

**new** ArrayList<>();

**private** **static** SortedMap<Integer,Employe> *mapEmploye*=

**new** TreeMap<Integer,Employe>();

// fonction permettant d’ajouter un Employe dans la liste

**public** **static** **void** ajouterEmployeListe(Employe employe) {

………………………………………………………………………………………………………………………………………

}

// fonction permettant de supprimer un Employe dans la liste

**public** **static** **boolean** supprimerEmployeListe(**int** matricule) {

………………………………………………………………………………………………………………………………………

}

// fonction permettant de renvoyer un Employe dans la liste

**public** **static** Employe renvoyerEmployeListe(**int** matricule){

………………………………………………………………………………………………………………………………………

}

// fonction permettant d’afficher les Employes de la liste

**public** **void** afficherEmployeListe(){

Iterator<Employe> it=…………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

}

// fonction permettant d’ajouter un Employe dans la map

// la clé = matricule

**public** **static** **void** ajouterEmployeMap(Employe employe) {

………………………………………………………………………………………………………………………………………

}

// fonction permettant de supprimer un Employe dans la map

**public** **static** **boolean** supprimerEmployeMap(**int** matricule){

………………………………………………………………………………………………………………………………………

}

// fonction permettant de renvoyer le premier Employe dans la // map

**public** **static** Employe renvoyerPremierEmployeMap(**int** matricule){

………………………………………………………………………………………………………………………………………

}

// fonction permettant d’afficher les Employes et leur

// matricule de la map

**public** **void** afficherMaptriculeEmployeMap(){

Set<Map.Entry<Integer,Employe>> set = ………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………… }

}

**//------------------------------------------------//**

Soit la classe suivante :

**public** **class** ExpressionLambda {

**private** **static** List<Employe> *liste* = Arrays.*asList*(

**new** Employe("Charkaoui", "Mouna", 2354, 15000, Departement.***ACHAT***),

**new** Employe("Erraji", "Zineb", 7685, 25000, Departement.***INFORMATIQUE***),

**new** Employe("Naji", "Mohammed", 3097, 10000, Departement.***RH***),

**new** Employe("Kamri", "Houda", 1209, 30000, Departement.***INFORMATIQUE***),

**new** Employe("Alaoui", "Adil", 5674, 35000, Departement.***COMMERCIAL***),

**new** Employe("Mchari", "Karime", 2046, 2300, Departement.***INFORMATIQUE***));

// function permettant de filtrer les employés suivant une

// condition

**public** **static** List<Employe> filtrerEmployes(Predicate<Employe> predicate){

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

}

**public** **static** **void** main(String[] args) {

// à l’aide des classes anonymes, implémenter un Comparator

Comparator<Employe> comp1 = …………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………

// à l’aide des expressions lambdas, implémenter un Comparator

Comparator<Employe> comp2= …………………………………………………………………………

………………………………………………………………………………………………………………………………………

// à l’aide des expressions lambdas, récupérer les employés

// appartenant au département informatique

List<Employe> employesInformatique = ……………………………………………………

………………………………………………………………………………………………………………………………………

// à l’aide des expressions lambdas, afficher ces employés

employesInformatique.forEach(…………………………………………………………………………………);

// à l’aide des expressions lambdas, récupérer les employés

// ayant un salaire supérieur ou égal à 2000

List<Employe> employesSalaireSup2000 =………………………………………………

………………………………………………………………………………………………………………………………………

// à l’aide des expressions lambdas, afficher ces employés

employesSalaireSup2000.forEach(……………………………………………………………………………);