TP N°7

Exercice 1 :

Un élève est caractérisé par les attributs privés suivants :

- nom : **`String`**,
- listeNotes **`: List<Double>`**, ensemble des notes obtenues par l'élève. Utilisez une **ArrayList** pour stocker les notes de l'élève.
- moyenne : **`double`.** Un élève sans aucune note aura une moyenne égale à 0.

➔ Soit la classe **Eleve** suivante, complétez par le code qui manque :

```
public class Eleve {

    private String nom;
    private List<Double> listeNotes = ……..;
    private double moyenne;

// Un constructeur permettant d'initialiser le nom de l'élève.

    public Eleve(String nom) {
        this.nom = …… ;
        this.moyenne = ……;
    }

//Un getter qui remplit l'attribut moyenne par la moyenne de l'élève et retourne sa valeur.

    public double getMoyenne() {

……
……
……
……
……
……
```

```
    }
//Un getter qui renvoie le nom de l'élève
    public String …… {
        return …….;
    }

 // Un getter qui renvoie la liste des notes de l'élève.

    public ……..getListeNotes() {
        return …….;
    }

 // Une méthode toString() qui retourne une description de l'élève sous la forme :

 // nomElève (moyenne), par exemple : Ali (12.25)

    @Override
    public ……. toString() {
        return …….. + " (" + String.format("%.2f", ……) + ")";
    }

// une méthode qui ajoute la note reçue en argument à la liste des notes ; si la note reçue est
//négative, la note ajoutée est 0 ;

    public void ajouterNote(double note) {

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



    }
}
```

➔ Soit une classe **TriParMoyenneComparator** qui implémente l'interface **Comparator** pour trier les élèves par moyenne croissante. Complétez par le code manquant :

```
public class TriParMoyenneComparator …… {
    @Override
    public int compare(…… e1, …… e2) {
        return …… ;
    }
}
```

- Soit la classe **`GroupeEleves`** ayant un attribut de type **List<Eleve>**. Complétez par le code qui manque :

```
public class GroupeEleves {

// Utilisez une LinkedList pour stocker les élèves du groupe.
  private List<Eleve> listeEleves = …….. ;
// une méthode qui renvoie le nombre d'élèves du groupe.
   public int nombre() {
      ……
   }
// une méthode qui ajoute un élève au groupe.
   public void ajouterEleve(Eleve eleve) {
      ……
   }
// une méthode qui recherche l'élève dont le nom est indiqué par le paramètre. La méthode renvoie le
//premier élève ayant ce nom sinon elle retourne null.
   public Eleve chercher(String nom) {
      ……..
      ……..
```

```
    ……..
    ……..
    ……..
    ……..

  }

// Une méthode qui affiche les élèves du groupe triés par ordre croissant selon la moyenne.

  public void lister() {
      //On crée une copie de listeEleves
      List<Eleve> L=new ArrayList<>(List.copyOf(listeEleves));
      //On trie la liste en ordre croissant selon la moyenne
       …… ;
      // On affiche la liste des élèves après le tri
      ……
      ……
       ……

  }
}
```

- Définir la classe **`GestionGroupes`** contenant une méthode **main** qui :
  a. Crée un groupe d'élèves.
  b. Ajoute des élèves au groupe.
  c. Affiche les élèves triés par ordre décroissant selon la moyenne.

Exercice 2 :

Un club sportif souhaite gérer ses joueurs de basketball. Un joueur est caractérisé par un nom, une année d'adhésion au club et une taille.

➔ Soit la classe **Joueur** qui implémente l'interface **Comparable**. On souhaite que les joueurs puissent être triées par nom puis par taille s'ils ont même nom. Complétez par le code manquant :

```
public class Joueur ……{
        private String nom;
        private int annee;
        private double taille;
        public Joueur(String nom, int annee, double taille) {
                this.nom =  …… ;
                this.annee = ……;
                this.taille = ……;
        }
        @Override
        public String toString() {
          return …… + "(" +String.format("%.2f", ……)+ ")" + ":" + ……..;
        }
        @Override
        public int compareTo(Joueur o) {
                int result = ……;
                if(result == 0)
                        return ……;
                return result;
        }
        public int getAnnee() {
                return …… ;
        }
}
```

➔ Soit une classe **TriAnneeComparator** qui implémente l'interface **Comparator.** On souhaite trier les joueurs par année d'adhésion décroissante. Cela permettra d'afficher les nouveaux joueurs en premier. Complétez par le code manquant :

```
public class TriAnneeComparator …….. {
    @Override
    public int compare(…….. o1, …… o2) {
        return …… ;
```

```
        }

    }
```

➔ Complétez le programme suivant par ce qui est demandé :

```
public class Club {
public static void main(String[] args) {

//Créez une ArrayList de joueurs.

        List<……> team=new …….. ;

//Remplir la liste par trois joueurs.

        team.add( new……..);

        …… ;

        …… ;

//Triez la liste selon l'ordre naturel (défini par l'implémentation de l'interface Comparable)

        ……

        for( Joueur J : team )
            System.out.println(j));

//Triez la liste par année d'adhésion décroissante à l'aide du comparateur TriAnneeComparator.
        ……

//Créez un TreeSet de joueurs. Le remplir par les éléments de la liste team.

      Set<……> setTeam =…… ;

// Affichez les joueurs de l'ensemble.

        ……

        ……

        }
      }
```

Exercice 3 : Qu'affiche le programme suivant :

```
Map<String, String> capitals = new TreeMap<>();
capitals.put("USA", "Washington D.C.");
capitals.put("France", "Paris");
capitals.put("Japan", "Tokyo");
for (String country : capitals.keySet()) {
System.out.println(country + ": " + capitals.get(country));
}
```

……

……

……

……

……

……

Exercice 4 :

On souhaite compter le nombre d'occurrences de chaque mot dans une phrase et stocker le résultat sous forme d'un HashMap (mot -> nombre d'occurrences). Ecrire le programme correspondant.