# Les collections

## Objectif(s)

- ✯ comprendre les différences entre liste, queue prioritaire, ensemble utilisant une table de hachage
- ✯ écrire un comparateur, des méthodes `compareTo()`, `hashCode()`
- ✯ implanter une file

## Exercice 1

Considèrez la classe `Etudiant` :

```
public class Etudiant {

        private static int etuPass = 21800001;

        private String formation;
        private String nom;
        private String prenom;
        private double noteMoyenneUE1;
        private double noteMoyenneUE2;
        private int ID;

        public Etudiant(String formation, String nom, String prenom) {
                this.formation = formation;
                this.nom = nom;
                this.prenom = prenom;
                noteMoyenneUE1 = 0.0;
                noteMoyenneUE2 = 0.0;
                ID = etuPass;
                etuPass++;
        }

        public void setNoteMoyenneUE(int unite, int [] notesUE){

                double somme = 0.0;

                if(notesUE.length > 0){
                        somme = notesUE[0];
                        for(int n = 1; n < notesUE.length; n++){
                                somme += notesUE[n];
                        }
                        somme /= notesUE.length;
                }

                if(unite == 1){
                        this.noteMoyenneUE1 = somme;
                }else if(unite == 2){
                        this.noteMoyenneUE2 = somme;
                }
```

```
        }

        public String toString(){
                return this.ID+": "+this.nom+", "+this.prenom+", Étudiant
                 ↪ "+this.formation+", UE1: "+this.noteMoyenneUE1+", UE2:
                 ↪ "+this.noteMoyenneUE2;
        }

}
```

*Question 1*

1. Que fait le programme ci-dessous ?
2. Est-ce qu'il y a des erreurs de compilation et d'exécution ?
3. Quel est le résultat du premier `System.out.println(...)` ? Quel est le résultat du deuxième et du troisième ?

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.PriorityQueue;


public class CollectionDEtudiant {

        public static void main(String[] args) {

                LinkedList<Etudiant> listeDEtudiants = new
                 ↪ LinkedList<Etudiant>();

                listeDEtudiants.add(new Etudiant("RT", "Alban", "Fourrier"));
                listeDEtudiants.add(new Etudiant("INFO", "Bert", "Germain"));
                listeDEtudiants.add(new Etudiant("RT", "Alban", "Fourrier"));
                listeDEtudiants.add(2, new Etudiant("INFOCOM", "Sophie",
                 ↪ "Herzig"));

                System.out.println(listeDEtudiants);

                HashSet<Etudiant> ensembleDEtudiants = new
                 ↪ HashSet<Etudiant>(listeDEtudiants);

                System.out.print(ensembleDEtudiants);

                PriorityQueue<Etudiant> pqDEtudiants = new
                 ↪ PriorityQueue<Etudiant>(ensembleDEtudiants);

                System.out.print(pqDEtudiants);
        }

}
```

*Question 2*

Ajouter le code nécessaire à la classe `Etudiant` pour qu'elle puisse être utilisé dans la `PriorityQueue`. La comparaison se fait en fonction de la ID de l'étudiant.

Après ce corrigé, quel est le résultat du troisième `System.out.println()` ?

*Question 3*

1. Écrivez un comparateur qui évalue l'ordre des deux étudiants dans le sens suivant :
   (a) En fonction du nom
   (b) Si les deux sont égaux, en fonction du prenom
   (c) Si les deux sont égaux, en fonction de la formation

   On suppose que toutes les méthodes d'accès aux attributs sont définis.
2. On ajoute quelques lignes au programme :

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

        ...

        listeDEtudiants.add(new Etudiant("INFO", "Agnes", "Nutter"));

        pqDEtudiants = new PriorityQueue<Etudiant>(new
          ↪ StringComparator());
        for(Etudiant e : listeDEtudiants){
                pqDEtudiants.add(e);
        }

        while(!pqDEtudiants.isEmpty()){
                System.out.println(pqDEtudiants.remove());
        }

        ...

}
```

   Dans quel ordre sont les étudiants affichés ?
3. Pourquoi on n'utilise pas simplement `System.out.println(pqDEtudiants)` ?

*Question 4*

1. Écrivez la méthode `equals()` de la classe `Etudiant` qui **ignore** l'attribut `ID` !
2. Comment ça change le comportement du programme ?
3. Écrivez la méthode `hashCode()` qui utilise seulement les attributs de type `String`.
4. Comment ça change le comportement du programme ? Et pourquoi ?
5. Pourquoi il y a un conflit entre l'Implémentation de la méthode `compareTo()`, de la méthode `equals()`, et de la méthode `hashCode()` ?

## Exercice 2 – Implémentation d'une file

Une file ou queue *(*FIFO : First In First Out), est une liste linéaire L pour laquelle les insertions sont réalisées à la fin de L (queue) et les suppressions sont effectuées au début de L (tête).

L'objectif de l'exercice est d'implanter de une file disposant de l'interface `File`.

```
interface File {
    /** Retourne si la file est vide */
    public boolean estVide();
    /** Ajoute un élément en queue de file */
    public void entrer(Object o);
    /** Retourne l'élément en tête de file */
    public Object sortir();
}
```

*Question 1*

On va implanter une file en utilisant un tableau. La classe s'appelera `FileTableau`. La représentation contiguë en tableau consiste à utiliser un tableau de taille fixe `N`, géré en de manière circulaire (buffer circulaire). Le suivant de l'indice `N-1` est `0` (utiliser deux indices : `iRemplissage` et `iVidage`). Cette technique permet d'éviter de faire des décalage dans les tableaux soit pour l'arrivée d'un élément dans la file, soit pour le retrait.

Écrire la classe `FileTableau` qui implémente l'interface `File`.