CONTROLE N°4

NOM

PRENOM

GROUPE

Exercice 1. Questions de Cours

a : Plus long plateau (2 points)

Écrivez le code de la méthode `plusLongPlateau` prenant en paramètre une liste de String et renvoyant la longueur du plus long plateau de cette liste.

Par exemple :

- Avec la liste `["a","a", "a", "b", "b", "c", "c", "a", "d"]`, votre méthode doit retourner `3`
- Avec la liste `["a","a", "b", "b", "b", "c", "c", "a", "d"]`, votre méthode doit retourner `3`
- Avec la liste `["a","a", "a", "b", "b", "c", "c", "c", "c"]`, votre méthode doit retourner `4`
- Avec la liste vide, votre méthode doit retourner `0`

réponse

b : Correct ou non ? (2 points)

1. Parmi les lignes de code suivantes :

- **rayez** celles dont la syntaxe est incorrecte (erreur à la compilation à cette ligne) ;
- mettez un petit **coeur** (ou une fleur, ou une étoile) à côté de celles qui respectent les bonnes pratiques.

```
ArrayList<> liste = new ArrayList<>();
List<Integer> liste = new List<>();
List<Integer> liste = new List<Integer>();
List<Integer> liste = new ArrayList<Integer>();
List<Integer> liste = new ArrayList<>();
List<> liste = new ArrayList<Double>();
List<> liste = new List<Double>();
```

2. Parmi les extraits de code suivants, rayez ceux dont la syntaxe est incorrecte :

```
public class Machin<T> implements Iterable<>
{ ... }
```

```
public class Machin<T> implements Iterable<T>
{ ... }
```

```
public class Machin implements Iterable<T>
{ ...}
```

```
public class Machin implements Iterable<>
{ ... }
```

```
public class Machin{
   public static void machin(T truc)
   { ... }
}
```

```
public class Machin{
   public static <T> void machin(T truc)
   { ... }
}
```

```
public interface Chose {
   public int machin(){ return 5; }
}
```

```
public interface Chose {
   public int machin(){}
}
```

```
public interface Chose {
   public int machin();
}
```

section*{Parcours Sup 2048}

En 2048, suite au perfectionnement des algorithmes en intelligence artificielle, les hommes ne travaillent plus et en sont arrivés à ne plus avoir besoin d'étudier les matières que nous connaissons aujourd'hui.

La fin de ParcoursSup ? Non. Et la sélection reste drastique.

Dans ce devoir, nous allons modéliser l'outil ParcoursSup2048

Exercice 2. La classe Etudiant

On modélise les futurs étudiants de l'année 2048 de la façon suivante :

- Un étudiant est identifié de manière *unique* par son nom et prénom. (Deux étudiants distincts ne peuvent avoir les mêmes noms et prénoms)
- Au lycée, les seules matières étudiées sont liées à la *parapsychologie*, le reste étant l'apanage des machines. Un futur étudiant a donc trois notes : l'une en **télépathie**, une autre en **précognition** (connaissance du futur) et la dernière en **télékinésie** (capacité à faire bouger les objets par la pensée, entre autres). Dans un soucis de simplification, nous supposerons que ces trois notes sont des notes *entières*.

1. (2 points) Écrivez une classe `Etudiant` contenant :
   - Un constructeur prenant deux paramètres : le nom et le prénom de l'étudiant.
   - Un constructeur prenant 5 paramètres : nom, prénom ainsi que les notes en télépathie, précognition et télékinésie.
   - Un observateur `getTelepathie`
   - Un setter `setPrecognition`

Pour vous aider, voici un exemple d'Executable :

```
class Executable {
     public static void main(String [] args) {
          Etudiant luke = new Etudiant("Luke", "Skywalker", 2, 8, 14);
          System.out.println(luke);
          // Luke Skywalker - Notes : télépathie=2 précognition=8 télékinésie=14
          Etudiant leia = new Etudiant("Leia", "Organa");
          System.out.println(leia);
          // Leia Organa - Il manque des notes
          leia.setTelepathie(15);
          System.out.println(leia);
          // Leia Organa - Il manque des notes
          leia.setTelekinesie(12);
          leia.setPrecognition(17);
          System.out.println(leia);
          // Leia Organa - Notes : télépathie=15 précognition=17 télékinésie=12
     }
}
```

Dans la suite du devoir, on supposera que les méthodes `getPrecognition`, `getTelekinesie`, `setTelepathie` et `setTelekinesie` sont également disponibles. *On ne vous demande pas de donner leur code*.

2. (3,5 points) Ajoutez à cette classe les trois méthodes issues de la classe `Object` que l'on vous demande habituellement d'implémenter.
3. (1,5 points) Complétez le code de façon à ce que les étudiants soient naturellement comparables par note en télékinésie **décroissante**. (C'est-à-dire etudiant1 est supérieur à etudiant2 si sa note en télékinésie est inférieure à celle de étudiant2)

Exercice 3. Modélisation des établissements

a : L'interface Établissement (1 point)

Dans notre modèle, un établissement d'enseignement supérieur est une entité capable de **sélectionner les étudiants**. On le modélisera par une interface `Etablissement` contenant les deux méthodes suivantes :

- `getSelection` prenant en paramètre une liste d'étudiants et renvoyant la liste triée par l'établissement (dans un ordre qui dépendra en fait de l'établissement)

- `getNbPlaces` ne prenant aucun paramètre et renvoyant le nombre de places disponibles.

Écrivez le code de cette interface

b : Classe École de BTP (1 point)

Une école de BTP est un établissement d'enseignement supérieur qui sélectionne ses futurs étudiants en fonction de leur note en télékinesie (tri par ordre de note de télékinésie **décroissante**).

Le nombre de places d'une telle école est passé en paramètre du constructeur.

Écrivez le code de cette classe.

c : Classe École de Politique (2 points)

Une telle école sélectionne selon la valeur de (télépathie + 0.5 * Précognition) **décroissante**. Il y a seulement 1 place : l'étudiant qui aura la chance d'intégrer cette école transmettra alors par télépathie l'ensemble des idées politiques ayant cours à cette époque.

Écrivez le code de cette classe.

Pour vous aider, voici un exemple d'Executable :

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

     // ...

     Etudiant solo = new Etudiant("Han", "Solo", 15, 10, 10);
     Etudiant chewie = new Etudiant("Chewbaka", "WaoWaaoWaooo", 2, 11, 18);
     Etudiant yoda = new Etudiant("Maitre", "Yoda", 13, 15, 20);
     List<Etudiant> listeEtudiants = Arrays.asList(luke, solo, chewie, yoda);

     Etablissement ecoleDesMines = new EcoleBTP("Ecole des Mines",3);
     System.out.println(ecoleDesMines.getSelection(listeEtudiants));
     //[(Maitre Yoda : télépathie = 13 précognition = 15 télékinésie = 20),
     // (Chewbaka WaoWaaoWaooo : télépathie = 2 précognition = 11 télékinésie =␣
↪18),
     // (Luke Skywalker : télépathie = 2 précognition = 8 télékinésie = 14),
     // (Han Solo : télépathie = 15 précognition = 10 télékinésie = 10)]

     Etablissement sciencePo = new EcolePolitique("Science Po");
     System.out.println(sciencePo.getSelection(listeEtudiants));
     // [(Maitre Yoda : télépathie = 13 précognition = 15 télékinésie = 20),
     // (Han Solo : télépathie = 15 précognition = 10 télékinésie = 10),
     // (Chewbaka WaoWaaoWaooo : télépathie = 2 précognition = 11 télékinésie =␣
↪18),
     // (Luke Skywalker : télépathie = 2 précognition = 8 télékinésie = 14)]
}
```

Exercice 4. Bibliothèque Attribution

Dans cet exercice, on vous demande d'écrire une bibliothèque **Attribution** avec les méthodes suivantes :

a : La méthode listePrincipale (2 points)

Écrire une méthode `listePrincipale` qui prend en paramètre une liste d'étudiants et une liste d'établissements et renvoie le dictionnaire qui à chaque étudiant associe les établissements qui les classent en liste principale (c'est-à-dire les étudiants dont le rang est inférieur au nombre de places disponibles dans l'école).

On applique l'algorithme suivant :

Pour chaque établissement :

- trier la liste des étudiants selon le critère de tri de l'établissement
- Pour i allant de 0 au nombre de places disponibles :
  - récupérer l'étudiant qui se trouve à la position i dans la liste triée
  - si l'étudiant n'est pas dans le dictionnaire, lui associer une liste vide
  - (dans tous les cas) ajouter l'établissement à la liste associée à l'étudiant dans le dictionnaire.

Pour vous aider, voici un exemple d'Executable :

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

        // ...

        List<Etablissement> lesEcoles = Arrays.asList(ecoleDesMines, sciencePo);
        System.out.println(Attribution.listePrincipale(listeEtudiants,lesEcoles));
        // {(Chewbaka WaoWaaoWaooo ... ) = [Ecole des Mines],
        // (Luke Skywalker ... ) = [Ecole des Mines],
        // (Maitre Yoda ... ) = [Ecole des Mines, Science Po]}
```

b : Deux méthodes max (2 points)

1. Sans utiliser la bibliothèque `Collections`, écrire une méthode `max1` qui prend en entrée une liste d'étudiants et renvoie l'étudiant qui a la meilleure note en télépathie. Cette méthode devra lever une exception si un tel étudiant n'existe pas.
2. En utilisant les itérateurs, écrire une méthode `max2` qui prend en entrée une liste d'étudiants et renvoie l'étudiant qui a la meilleure note en télépathie. Cette méthode devra lever une exception si un tel étudiant n'existe pas.

c : Écrire une classe Itérable (2 points)

On vous donne le code d'une classe `BonsEnTelepathie` qui implémente `Iterable<Etudiant>`.

```
class BonsEnTelepathie implements Iterable<Etudiant>  {
    private List<Etudiant> etudiants;
    private int seuil;
    // seuil au dessus duquel les étudiants sont "bons"
    public BonsEnTelepathie(List<Etudiant> etudiants, int seuil) {
        this.etudiants = etudiants;
        this.seuil = seuil;
    }
    public BonsEnTelepathie(double seuil) {
        this(new ArrayList<>(), seuil);
    }
    public void ajouteEtudiant(Etudiant etudiant) {
        this.etudiants.add(etudiant);
    }
    public void setSeuil(int seuil){
        this.seuil = seuil;
    }
    @Override
    public Iterator<Etudiant> iterator() {
        return new IterEtudiants(etudiants, seuil);
    }
}
```

On vous demande d'implémenter l'itérateur `iterator<Etudiant>` permettant d'assurer le parcours avec un *foreach* comme par exemple dans l'Executable suivant. Le constructeur de cet itérateur prend en entrée un entier et une liste. La méthode next de cet itérateur renverra les étudiants dont la note en télépathie est supérieur à cet entier, en « sautant » ceux dont la note est inférieure à ce seuil.

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

        // ...

        BonsEnTelepathie lesBons = new BonsEnTelepathie(listeEtudiants, 2);
        for(Etudiant etudiant : lesBons){
            System.out.println(etudiant);
        }
        // (Han Solo : télépathie = 15 précognition = 10 télékinésie = 10)
        // (Maitre Yoda : télépathie = 13 précognition = 15 télékinésie = 20)
    }
```