# UEF4.3. Programmation Orientée Objet

# EXAMEN FINAL- corrigé

**02 heures - Documents interdits**

**Exercice 1 (9 points)/**

a. Cas1: les données météo d'une wilaya sont triées par ordre chronologique
Solution: 1pt
private Collection<Weather> dailyWeather
= new TreeSet(); 0.5
Pour le bon fonctionnement du programme il faut que la classe Weather implémente l'interface Comparable et redéfinisse la méthode compareTo de façon à comparer les dates comme suit

```
public class Weather implements
        Comparable<Weather>0.25{
    .....

     // Cas a: les données météo
        sont triées par ordre
        chronologique
     public int
        compareTo(Weather o)0.25{
         return
        date.compareTo(o.date);
     }
```

b. Cas2: les données météo d'une wilaya sont triées selon l'ordre décroissant de la température 1pt
les conditions sont identiques au cas a à la différence que la méthode 0.5 compareTo doit être redéfinie comme suit:

```
public int compareTo(Weather o){0.5
 if (temp- o.temp<0)return 1;
 else if(temp- o.temp>0) return -1;
        else return
date.compareTo(o.date);}
```

2.

a. Cas 1 : les wilayas sont stockées par ordre alphabétique de leurs noms
solution : 1pt
ligne 2: **`Map<String,Wilaya> WilayasWeather;`** 0.5
ligne 3: **`WilayasWeather = new TreeMap();`**0.5
Rien d'autre n'est nécessaire puisque dans la classe String compareTo est déjà redéfini

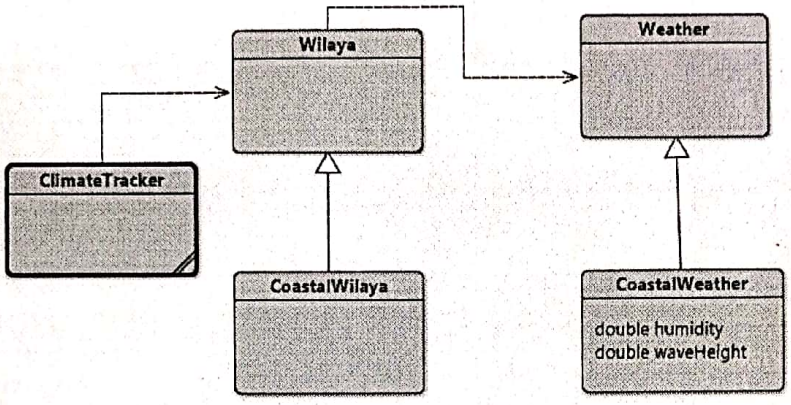
b. Cas 2 : les wilayas sont stockées par ordre croissant de la moyenne de pluviométrie (la moyenne est calculée sur l'ensemble des valeurs stockées jusqu'à ce jour) et à chaque Wilaya est associée la moyenne de sa pluviométrie
solution : 2pts
ligne 2 : `Map<Wilaya,Double> WilayasWeather;` 0.5
ligne 3 : `WilayasWeather = new TreeMap<Wilaya,Double>();`0.25
Pour le bon fonctionnement du programme il faut:
La classe Wilaya doit implémenter comparable de façon à trier selon l'ordre croissant de la moyenne de la pluviométrie

```
public class Wilaya implements
Comparable<Wilaya> 0.25{
// attributs
public double
getAverageRainfall(){ 0.5
    double sum = 0;
    for(Weather w : dailyWeather)
        sum =sum +
w.getRainfall();
        return
sum/(dailyWeather.size());
}

public int compareTo(Wilaya w){0.5
    double dif =
getAverageRainfall()-
w.getAverageRainfall();
    if (dif<0)return -1;
     else if(dif>0) return 1;
        else return
nom.compareTo(w.nom);;
}
```

3. L'association veut développer *ClimatTracker* pour observer de plus près les wilayas côtières qui se caractérisent par le taux d'humidité et la hauteur des vagues. Proposez une solution pour répondre à ce besoin de façon à apporter le moins de modifications aux classes existantes.
   a. Tracer le nouveau diagramme de classes en indiquant les attributs et méthodes des nouvelles classes. 1,5

solution: il suffit d'ajouter une classe **CoastalWilaya dérivant de la classe Wilaya** 0.5 et une classe **CoastalWeather dérivant de la la classe Weather** 0.5 ayant **deux nouveaux attributs humidity et waveHeught de type double.** 0.5




b- Expliquez ce qui a permis de faire évoluer votre programme avec le moins de modifications possibles en précisant les principes OO utilisés: **1,5pt**
Cette solution ne nécessite aucune modification au niveau des classes Weather, Wilaya et Climate car nous avons exploité les principes d'héritage et de polymorphisme.
Explication:

- La classe Weather elle est juste dérivée: principe d'héritage 0.25
- La classe Wilaya est aussi dérivée héritage 0.25 pour créer la classe CoastalWilaya et ne nécessite aucune modification car son attribut de type Weather peut recevoir une référence de type CoastalWeather: principe de polymorphisme.0.5
- La classe ClimateTracker ne nécessite aucune modification car sa collection de type Map peut contenir aussi bien des objets de type Wilaya que CoastalWilaya: principe de polymorphisme. 0.5

4. Pour sauvegarder les données entre deux exécutions du programme, on peut utiliser un fichier texte contenant une ligne consacrée à chaque wilaya.
   a. Expliquez comment une telle solution peut être implémentée. 0.5
   Il suffit d'adopter un format particulier pour le stockage des données pour pouvoir écrire les données et les récupérer en respectant le format.
   Exemple
   NomWilaya, date1, temp1,wind1,rainfall1, date2, temp2,wind2,rainfall2,

   b. Il existe un moyen plus simple et plus efficace pour assurer cette fonctionnalité. Décrivez-le en citant les classes java nécessaires pour l'implémenter.

On peut utiliser la **sérialisation qui consiste à rendre les objets persistants en les écrivants dans un fichiers** 0.25 en utilisant les classe ObjectInputStream et ObjectOutputStream 0.25

## Exercice 2. ( 5.5 pts)

1. Donner le programme Java de la classe Pile qui sert à empiler des entiers et comportant un constructeur, une méthode empiler et une méthode dépiler qui lance une exception si la pile est vide.

Sol **2 pts**

**Remarque: cette question ayant été traitée en TD, les signatures des méthodes n'ont pas été notées. Une signature erronée fait perdre le 0.25**

```
class Pile {
private Deque<Integer>pile;
  //0,25 accepter linkedList aussi
//constructeur
public  Pile(i){
  pile=new ArrayDeque<Integer>();
//0,25 accepter linkedList, le type
d'élement doit être Integer pas int
}
  //  empile x au sommet de la pile
public void empiler(int x){
  pile.push(x);
//0,25 accepter addFirst aussi
}
public int depiler() throws
PileVideException{  //throw =  0,25
if (this.estVide()) throw new
PileVideException();  // test+  throw =
0,25
return pile.pop();    //0,25 accepter
getFirst aussi}

public boolean estVide() {
return pile.isEmpty();   //0,25}
}
class PileVideException extends
Exception{}   //0,25
```

2. Ecrire un programme permettant, à l'aide de cette pile, de convertir un nombre décimal *x* en un nombre écrit dans une base *n*. Le nombre et la base sont entrés par l'utilisateur et le programme affiche le résultat après la conversion.
3. Sol **3,5pts**

```
public class ex2CF{
   private Pile p;  // 0,25
   public ex2CF(){
       p = new Pile();  // 0,25
    }
   public int convertir(int x, int n) { // 0,75   logique correcte+empiler
      int y=0;
       while (x!=0){
      p.empiler(x % n) ;
            x=x/n;   }
 try{ // 0,25
          if (!p.estVide()){
  // 0,5  : logique correcte depiler+test pileVide
 y=p.depiler();
 while (!p.estVide()){
 y=y*10+p.depiler(); }
 }
 }
catch(PileVideException e){// 0,25
System.out.println("pile vide");
}
return  y;
 public static void main(String args[]) { System.out.println ("introduire le
nombre décimal à convertir :");
Scanner input = new Scanner(System.in); // 0,25       int x =
input.nextInt();    // 0,25
System.out.println ("introduire la base :");
input = new Scanner(System.in);
int n = input.nextInt();// 0,25
ex2CF m=new ex2CF();
int resultat=m.convertir(x,n);    //0,25
System.out.println ("Résultat :"+resultat);     // 0,25
  }
}
```

**Exercice 3 (5.5 pts) :**

**Question 1.** 2pts

Corrigé:

a. 0,5 (Citer le principe)

Le principe de polymorphisme.

b. 0,5*3=1,5pt

Les objets télécommandés doivent implémenter une interface comportant les méthodes attendues par la télécommande. ces méthodes sont implémentées différemment dans chacun des objets télécommandés

Les méthodes de la télécommande doivent accepter des objets de classes implémentant cette interrface (0,5pt).

Par exemple pour les méthodes suivant et précédent de la télécommande on doit avoir un code semblable à celui là dans la classe télécommande (les boutons associés): (0,5pt),

suivant(ObjetM o){o.suivant()}

precedent(ObjetM o){o.precedent()}

Dans la classe téléviseur, la méthode suivant (respectivement precedent) est définie de sorte à renvoyer vers la chaîne suivante (respectivement precedente) ,

dans la classe porte. la méthode suivant(respectivement precedent) renvoie vers l'état fermé ou ouvert selon l'état courant de la porte(0,5pt)

**Question 2.**

Corrigé: 3,5pts

a. type de classes: 2,25 pts

classe Personne classe simple(englobante) 0,25

la classe Comparateur: classe interne statique 0,25+0.5

Classe locale anonyme implémentant l'interface I. 0,25+0,5

classe Test classe simple(comportant la méthode principale) 0.25

Interface I 0.25

b. résultat d'exécution: 1,25pts

-4 (accepter n'importe quel nombre négatif) 0,5

Mourad Belkacem âge 63 // 0.25 pour noms+prénom et 0,5 pour l'âge