# 3. Feuille de TP

## semaine 13

### Article

Un article est reconnu par son nom, il a un prix et un poids.

1. Écrire la classe **Article** avec un constructeur, les getteurs et les méthodes usuelles de la classe Object.
2. On veut en plus que les articles soient *triables naturellement* en fonction de leur prix croissant, compléter la classe **Article**.

### Comptable

Définir une interface **Comptable** disposant de la méthode *combien()* retournant un entier.

### Rayon

1. Définir un **Rayon** qui a un nom, une liste d'article et qui est Comptable.
2. Ajouter dans **Rayon** une méthode *trierParPrix()* qui trie les articles par *prix croissant*.
3. Ajouter dans **Rayon** une méthode *quelArticle(String nom)* qui retourne l'article nommé nom, présent dans le Rayon (le premier trouvé) en gérant les potentielles exceptions.

### Bibliothèque

1. Écrire une bibliothèque **GestionRayon** qui implémente une méthode *double moyenneDesPrix(Rayon rayon)* qui retourne la moyenne des prix des articles présents dans le rayon. Penser aux exceptions.
2. Ajouter à votre biblothèque une méthode *List<Article> trierParNom(Rayon rayon)* qui retourne la liste d'articles contenue dans le rayon triée en fonction du *nom de chaque article*. Le rayon ne doit pas être modifié.

### Exécutable

Écrire un exécutable permettant de tester vos méthodes.

### Gérer les rendez-vous de mon Agenda

**Note:**
L'objectif est de créer un programme pour gérer des rendez-vous. Pour cela, vous devrez écrire deux classes :

- une classe **RendezVous** représentant la notion de rendez vous : une date de début et une date de fin ;
- une classe **Agenda** qui permettra de gérer nos rendez-vous.

### La classe Date de l'API Java

Pour gérer les dates, vous utiliserez la classe **Date** de l'API java. La classe **Date** implémente *Comparable<Date>* et peut donc *naturellement* être comparée. Voici un extrait de la documentation de deux méthodes utiles de cette classe :

**Method Detail**

**compareTo**

```
public int compareTo(Date anotherDate)
```

Returns:

the value 0 if the argument Date is equal to this Date; a value less than 0 if this Date is before the Date argument; and a value greater than 0 if this Date is after the Date argument.

**toString**

```
public String toString()
```

Converts this Date object to a String of the form:

```
dow mon dd hh:mm:ss zzz yyyy
```

Par ailleurs, on pourra convertir une chaine en **Date** à l'aide la méthode *parse* de **SimpleDateFormat** :

```
lecteur = new SimpleDateFormat("HH:mm/dd/MM/yyyy");
// précise le format de saisie de la date
String chaine = "10:15/05/01/2023"; // 10h15 le 5 janvier 2023
Date d = lecteur.parse(chaine);
// Attention, parse peut lever l'exception ParseException
// si la chaine n'a pas le format demandé.
```

Complétez le code suivant :

```
lecteur = new SimpleDateFormat("HH:dd/MM/YYYY");
String chaine1 = "10:00/05/12/2023"; // 10h00 le 5 décembre 2023
try
{
     Date d1 = lecteur.parse(chaine1);
     Date d2 = lecteur.parse("10:30/06/12/2023");
       if(                                              )
        {
           System.out.println(d1 + " est antérieur à " + d2 );
         }
        else
        {
           System.out.println(d1 + " est postérieur à " + d2 );
         }
}
catch(                                          )
{
```

(suite sur la page suivante)

(suite de la page précédente)

```
    System.out.println("Probleme dans le format d'une des deux dates");
}
```

### La classe RendezVous

Définissez la classe **RendezVous**. On veut que les RendezVous soient naturellement comparables par date de début croissante.

```
public class RendezVous
 {
     private Date debut;
     private Date fin;

     public RendezVous(String deb, String fin)                                {
         SimpleDateFormat lecteur = new SimpleDateFormat("HH:mm/dd/MM/yyyy");
         this.debut = lecteur.parse(deb);
         this.fin = lecteur.parse(fin);

     }
     @Override
     public String toString() {



     }

     @Override
     public int                          {



     }

     public boolean intersecte(RendezVous rendezVous)
     {
     // Renvoie vrai si this et rendezVous 'tombent en même temps'









     }
}
```

### La classe Agenda

Complétez la classe **Agenda** suivante :

```
public class Agenda
{

    private List<RendezVous> contenu;

    Agenda() {


    }

    public RendezVous getPremier()                    {
    // Renvoie le premier (première date de début) rendez-vous
    // de l'agenda s'il existe




    }

    public void ajoute (RendezVous rendezVous)
    // attention les rendez-vous doivent être valides!
    {







    }

    @Override
    public String toString()
    {


    }
}


...
```

### Un exécutable

Écrivez un exécutable avec :

- La création de cinq rendez-vous, dont deux qui s'intersectent. N'ajoutez que les rendez-vous 'valides'.
- L'affichage du premier rendez-vous (selon la date de début).
- Pensez à tester les divers cas d'erreur.

### La classe RendezVousCourt

- On souhaite créer une classe **RendezVousCourt** qui correspond à un rendez-vous de 5 minutes. Dans cette classe, on ne stocke donc que la date de début du rendez vous. Proposez un diagramme de classes permettant d'avoir dans l'agenda à la fois des rendez-vous avec l'implémentation précédente et des rendez-vous "courts" (Précisez ce qu'il faudrait ajouter et/ou modifier dans le code précédent).
- Modifier vos codes et ajoutez le nécessaire. Modifiez également l'exécutable.

**Note:**
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 tp, nous allons modéliser l'outil ParcoursSup2048

### 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. 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.
   - les observateurs et setters nécessaires.

Voici un exemple d'exécutable :

```
public 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
     }
}
```

2. Ajoutez à cette classe les trois méthodes issues de la classe `Object` que l'on vous demande habituellement d'implémenter.
3. Complétez le code de façon à ce que les étudiants soient naturellement comparables par note en télékinésie **décroissante**.

## Modélisation des établissements

### Établissement

Dans notre modèle, un établissement d'enseignement supérieur est une entité capable de **sélectionner les étudiants**. Modélisez cet établissement 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.

### Classe École de BTP

Une école de BTP est un établissement d'enseignement supérieur qui sélectionne ses futurs étudiants en fonction de leur note en télékinésie (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.

### Classe École de Politique

Une telle école sélectionne selon la valeur de (télépathie + 0.5 * Précognition) **décroissante**. Il n'y a qu'une 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 :

```
public 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, leia);

     Etablissement ecoleDesMines = new EcoleDeBTP("Ecole des Mines", 3);
     System.out.println(ecoleDesMines.getSelection(listeEtudiants));
     // Sélection de l'école des mines
     // [Han Solo - télépathie : 10 précognition : 10 telekinésie : 15,
     //  Maitre Yoda - télépathie : 15 précognition : 20 telekinésie : 13,
     //  Leia Organa - télépathie : 15 précognition : 17 telekinésie : 12,
     //  Luke Skywalker - télépathie : 8 précognition : 14 telekinésie : 2,
     //  Chewbaka WaoWaaoWaooo - télépathie : 11 précognition : 18 telekinésie : 2]

     Etablissement sciencePoParis = new EcolePolitique("Science Po Paris");
     System.out.println(sciencePoParis.getSelection(listeEtudiants));
     // Sélection de science po paris
     // [Maitre Yoda - télépathie : 15 précognition : 20 telekinésie : 13,
     //  Leia Organa - télépathie : 15 précognition : 17 telekinésie : 12,
     //  Chewbaka WaoWaaoWaooo - télépathie : 11 précognition : 18 telekinésie : 2,
     //  Luke Skywalker - télépathie : 8 précognition : 14 telekinésie : 2,
     //  Han Solo - télépathie : 10 précognition : 10 telekinésie : 15]
```

(suite sur la page suivante)

(suite de la page précédente)

```
    Etablissement ecoleDesBeauxArts = new EcoleDeBTP("Les beaux arts", 5);

    Etablissement sciencePoRennes = new EcolePolitique("Science Po Rennes");
    }
}
```

### Bibliothèque Attribution

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

### La méthode listePrincipale

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 un ensemble vide
  - (dans tous les cas) ajouter l'établissement à l'ensemble associé à l'étudiant dans le dictionnaire.

Pour vous aider, voici un exemple d'exécutable :

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

        // ...

        List<Etablissement> lesEcoles = Arrays.asList(ecoleDesMines, sciencePoParis,␣
↪ecoleDesBeauxArts, sciencePoRennes);
        System.out.println(Attribution.listePrincipale(listeEtudiants,lesEcoles));
        // Liste Principale
        // {Chewbaka WaoWaaoWaooo - télépathie : 11 précognition : 18 telekinésie : 2
        //  = [Les beaux arts (5 places)],
        //  Leia Organa - télépathie : 15 précognition : 17 telekinésie : 12
        //  = [Ecole des Mines (3 places), Les beaux arts (5 places)],
        //  Han Solo - télépathie : 10 précognition : 10 telekinésie : 15
        //  = [Ecole des Mines (3 places), Les beaux arts (5 places)],
        //  Luke Skywalker - télépathie : 8 précognition : 14 telekinésie : 2
        //  = [Les beaux arts (5 places)],
        // Maitre Yoda - télépathie : 15 précognition : 20 telekinésie : 13
        //  = [Ecole des Mines (3 places), Science Po Rennes (1 place),
        //     Science Po Paris (1 place), Les beaux arts (5 places)]}
```

### Méthode max

Sans utiliser la bibliothèque `Collections`, écrire une méthode `maximum` 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.