TD 2 — les premières classes

Objectif(s)

- ✯ définir des classes
- ✯ définir les différentes méthode d'objets
- ✯ définir des méthodes/attributs de classe

Exercice 1 – Classe Intervalle

Définir une classe `Intervalle` qui a deux attributs entiers (bornes min et max). On considère que les deux bornes sont incluses dans l'intervalle. Ces deux attributs sont privés. La classe est dotés des méthodes (et constructeurs) suivants :

1. Un constructeur créant par défaut un intervalle $[0, 0]$ et un autre prenant deux entiers (bornes supérieure et inférieure pouvant être fixées dans n'importe quel ordre).
2. (**À faire en TP**) Les méthodes d'accès aux attributs (getters).
3. Attention, on ne fera qu'un seul setter que l'on nommera `setMinMax(int, int)` afin d'assurer la cohérence de l'intervalle avec une valeur min qui est inférieure à la valeur max.
4. Une méthode `String toString()` retournant une chaîne de caractères de la forme : `[min ; max]`.
5. Une méthode `void afficher()` pour afficher l'intervalle. L'affichage est de la forme : " L'intervalle a pour borne inférieure `min` et pour borne supérieure `max` ".
6. Une méthode `int longueur()` retournant la longueur de l'intervalle.
7. Une méthode `boolean estIncluse(int)` retournant si une valeur est interne à l'intervalle (valeurs "vrai" ou "faux").
8. Une méthode permettant de tester l'égalité (`boolean equals(Intervalle)`) avec un autre intervalle.
9. Une méthode `boolean contient(Intervalle)` permettant de tester l'inclusion d'un intervalle à l'intérieur de l'intervalle considéré (pensez à utiliser les méthodes que vous avez déjà définies).
10. Une méthode `Intervalle intersection(Intervalle)` permettant de calculer l'intersection avec un autre intervalle. Le nouvel intervalle correspondant à cette intersection est **retourné**, si elle n'existe pas la méthode retourne `null`. L'objet courant **n'est pas** modifié.

Question 1

Définir l'ensemble de ces méthodes.

Question 2 – **À faire en TP**

Proposer un ensemble d'instructions Java permettant de tester le bon fonctionnement de ces méthodes en testant les cas particuliers :

— des intervalles de taille 0

— des intervalles qui ne se chévauchent pas

— etc.

Exercice 2 – La classe `Personne`

On veut définir une classe nommée `Personne` représentant certaines caractéristiques d'une personne. La déclaration d'une personne revient à définir son nom, son adresse, son âge et à préciser comment on accède aux informations. Le nom et l'adresse sont des chaînes de caractères, l'âge est un entier. La classe `Personne` dont l'état est défini par trois attributs privés (nom, adresse et âge) a quatre méthodes :

— Un constructeur qui permet de donner des valeurs par défaut aux attributs (“” aux chaînes de caractères, 0 à l'âge).

— Un constructeur qui prend trois paramètres (deux chaînes de caractères et un entier) et stocke leurs valeurs dans les attributs de la nouvelle instance.

— La méthode `String categorieDAge()` qui retourne une des chaînes de caractères suivantes "Personne jeune", "Personne d'âge moyen" ou "Personne âgée" selon la valeur de l'attribut âge.
La catégorie d'âge est définie comme suit : Si l'age est > 60 alors "Personne âgée", si l'age est < 20 alors "Personne jeune" et sinon "Personne d'âge moyen".

— La méthode `categorieDeResidence()` qui affiche "Habitant urbain", "Villageois" ou "Habitant rural" selon la valeur de l'attribut adresse. La catégorie d'adresse est définie comme suit : Si l'adresse contient le mot Ville alors "Habitant urbain", si elle contient le mot Village alors "Villageois" sinon "Habitant rural".

Question 1
Écrire les constructeurs, les deux méthodes `categorieDAge()` et `categorieDeResidence()`. Écrire également les méthodes permettant la représentation textuelle (`String toString()`) et la comparaison d'instances (`boolean equals()`) de la classe `Personne`.

Question 2
(**À faire en TP**) Dans le programme principal, créer trois personnes avec les propriétés suivantes : `("Louis", "Ville", 25)`, `("Emile", "Village", 72)`, `("Charles", "Campagne", 43)`.

Question 3
Créer un tableau de personnes et remplissez-le avec les instances déjà créées, puis appelez les méthodes `categorieDAge()` et `categorieDeResidence()` pour chacune des personnes du tableau après les avoir affichées.

Exercice 3 – classe `Vecteur3d`

La classe `Vecteur3D` a 3 attributs (privés) `x`, `y` et `z` de type `double`.

Question 1

1. (**À faire en TP**) Définir un constructeur par défaut (`x`, `y` et `z=0`) et un constructeur prenant 3 paramètres.
2. (**À faire en TP**) Définir les méthodes d'accès et de modification de ces trois attributs.
3. (**À faire en TP**) Redéfinir la méthode `String toString()` qui retourne une chaîne de caractères de la forme : “Vecteur3D(x,y,z)”, x, y, z étant remplacé par les valeurs des attributs de l'instance.
4. (**À faire en TP**) Redéfinir la méthode `boolean equals(Vecteur3D)` pour permettre la comparaison d'instance.
5. Définir la méthode :
   — `public void recopie(Vecteur3D v2)`,
   qui recopie l'ensemble des valeurs des attributs de `v2` dans ceux de l'instance de la classe `Vecteur3D` exploitant cette méthode.
6. Définir une méthode `public Vecteur3D clone()` qui crée une *nouvelle* instance de la classe `Vecteur3D` dont les attributs ont les mêmes valeurs que l'objet participant à cette création.
7. Définir maintenant une méthode (non statique) de prototype : `public double produitScalaire(Vecteur3D)`. Cette méthode d'objet utilise l'objet (this) comme premier élément du produit scalaire.
8. Définir maintenant la méthode d'instance :
   — `public Vecteur3D somme(Vecteur3D)` qui calcule la somme de vecteur appelant et du vecteur en paramètre.

9. Ajouter une méthode de classe dont le prototype est :

```
public static double produitScalaire(Vecteur3D, Vecteur3D)}
```

10. Définir les deux méthodes de classe :
    — `public static Vecteur3D somme( Vecteur3D, Vecteur3D)`
    — `public static Vecteur3D produitVectoriel(Vecteur3D, Vecteur3D)`
11. – **À faire en TP** Dans un nouveau fichier, créer une classe `testVecteur3D` avec une méthode `main` qui contiendra un ensemble d'instructions permettant de vérifier le bon fonctionnement de la classe vecteur3D à partir de la manipulation de plusieurs instances.

Question 2

On souhaite ajouter un compteur permettant de savoir combien d'instances de `vecteur3D` sont référencées à un moment de l'exécution d'un programme. Proposer les modifications nécessaires et fournir une méthode `int ↪ getNbVect()` qui renvoie le nombre de vecteurs référencés. Vous penserez fournir un mécanisme pour prendre en compte la suppression d'un vecteur par le garbage collector.