TP N°3 *Classes enveloppes, Listes et Collections*

Objectifs

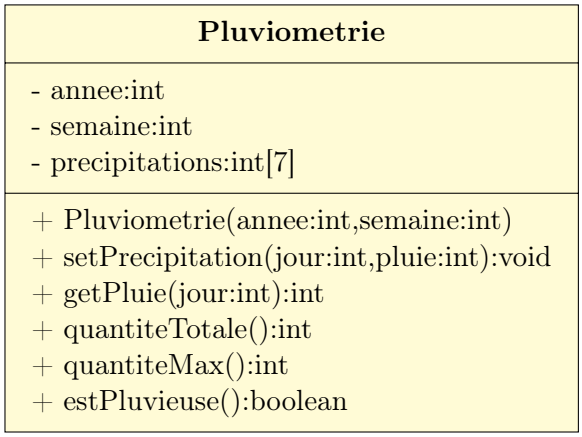
- JAVA : types primitifs et classes enveloppe
- JAVA : la méthode `equals()`

Exercice 1 *Pluviométrie*

Voici un extrait des données pluviométriques sur Orléans en 2022. Les précipitations sont données en mm et un "-" indique que la mesure n'est pas disponible.

	lundi	mardi	mercredi	jeudi	vendredi	samedi	dimanche
semaine 35	3	-	0	0	16	3	0
semaine 36	11	0	2	0	0	0	0
semaine 37	0	1	0	0	-	-	0
semaine 39	6	4	5	1	1	4	0

On veut développer une application qui permet d'analyser la pluviométrie sur une semaine. Pour cela, on décide d'utiliser une classe `Pluviometrie` dont on donne la représentation UML.




```java
public class Executable{
  public static void main(String[] args){
    Pluviometrie s35 =
        new Pluviometrie(2022, 35);
    s35.setPrecipitation(0, 3);
    s35.setPrecipitation(2, 0);
    s35.setPrecipitation(3, 0);
    s35.setPrecipitation(4, 16);
    s35.setPrecipitation(5, 3);
    s35.setPrecipitation(6, 0);
    System.out.println(s35.getPluie(1));
      //null
    System.out.println(s35.quantiteTotale());
      //22
    System.out.println(s35.quantiteMax());
      //16
    System.out.println(s35.estPluvieuse());
      //true
  }
}
```

Une semaine est considérée comme pluvieuse si il a plu pendant 2 jours consécutifs.

1.1 Un diagramme UML étant indépendant du langage, on n'a pas fait la distinction entre `int` et `Integer`. Sachant que ce projet doit être implémenté en Java, identifie les endroits où on peut utiliser un type primitif `int` et les endroits où il est nécessaire d'utiliser une classe enveloppe `Integer`

1.2 Recopie le code de la classe `Executable` puis écris le CODE MINIMAL de la classe `Pluviometrie` pour que le projet compile.

1.3 Complète le code du constructeur ainsi que le code de la méthode `setPrecipitation()`. Pour le moment les autres méthodes ne font RIEN.

1.4 Complète l'exécutable avec d'autres données et ajoute des tests pour chacune des méthodes à implémenter. Vérifie que tous tes tests ECHOUENT.

1.5 Complète le code du/des constructeur(s), puis écris le code de autres méthodes une à une. A chaque étape, compile, vérifie que les « pseudo-tests » qui sont dans l'exécutable passent.

Exercice 2 *Modéliser une cave à vin*

Dans cet exercice, on va chercher à modéliser une cave à vin. Pour cela, on crée une classe `Bouteille` permettant de modéliser une bouteille. On doit pouvoir exécuter le code suivant (tu peux ajouter des tests)

```
public class ExecutableBouteille{
    public static void main(String[] args){
        Bouteille a = new Bouteille("Bordeaux", "Pomerol", 2007);
        assert "Bordeaux".equals(a.getRegion());
        assert "Pomerol".equals(a.getAppellation());
        assert 2007 == a.getMillesime());
        Bouteille b = new Bouteille("Bordeaux", "Pomerol", 2007);
        Bouteille c = new Bouteille("Bordeaux", "Pomerol", 2003);
        assert a.equals(b);
        assert !a.equals(c);
    }
}
```

2.1 Écris le code de cette classe et vérifie que les pseudo-tests de l'exécutable passent.

On va maintenant modéliser une cave à l'aide d'une classe `Cave` [1]. On veut pouvoir exécuter le code suivant :

```
public class ExecutableCave{
    public static void main(String[] args){
      // reprendre le code de la question précédente
      Cave maCave = new Cave();
      maCave.ajouteBouteille("Bordeaux", "Pomerol", 2005);
      maCave.ajouteBouteille("Bordeaux", "Pomerol", 2007);
      maCave.ajouteBouteille("Bourgogne", "Nuits St George", 2001);
      maCave.ajouteBouteille("Savoie", "Pinot Noir", 2012);
      maCave.ajouteBouteille("Bordeaux", "Pomerol", 2007);
      maCave.ajouteBouteille("Loire", "Chinon", 2017);
      assert 6 == maCave.nbBouteilles();
      assert 3 == maCave.nbBouteillesDeRegion("Bordeaux");
      Bouteille b = maCave.plusVieilleBouteille();
      assert "Nuits St George".equals(b.getAppellation());
      assert maCave.contient("Bordeaux", "Pomerol", 2007);
      assert !maCave.contient("Bordeaux", "Pomerol", 2003);
}
```

2.2 Écris le code minimal de la classe `Cave` pour que le projet compile. Pour le moment ton code ne fait RIEN et les tests doivent ÉCHOUER.

2.3 Complète le code du/des constructeur(s), puis écris le code de autres méthodes une à une. A chaque étape, compile, vérifie que les « pseudo-tests » qui sont dans l'exécutable passent. Tu peux compléter l'exécutable pour ajouter des tests.

1. Tu noteras l'extrême originalité du concepteur de cet exercice pour le choix du nom de ses classes !