# TP de PPO en JAVA

Polytech Lille GIS2A3

**Objectifs** factorisation de code, classes abstraites et polymorphisme.

## 1 Progressions arithmétiques

Créer un répertoire `tp3` puis un répertoire de travail de nom `arithmetic` pour les classes de cette section.

Programmer une classe `ProgressionArithmetique` comme suit. Une progression arithmétique est caractérisée par un *premier terme* et une *raison* (`double`), paramètres du constructeur. Les termes sont mémorisés dans une `ArrayList` de `Double`, le dernier terme calculé est repéré par son indice, soit `rang`. Les termes sont calculés par appels successifs à une méthode `'void next()'` qui fait progresser l'indice `rang` de 1 et mémorise le terme correspondant. La méthode surchargée `'void next(n)'` permet de calculer `n` termes suivants par itération de `next()`. Une méthode `'double getTerme()'` permet d'obtenir le dernier terme calculé (rangé à l'indice `rang`). Enfin la méthode `toString()` renvoie une chaine de caractères formée des termes de la progression séparés par des espaces.

Programmer une classe principale `TestArithmetique` qui permet de créer et manipuler une progression arithmétique comme suit :

```
> java TestArithmetique
premier terme? 10
raison? 2
next (y/n)? y
-> 12.0
next (y/n)? y
-> 14.0
next (y/n)? y
-> 16.0
next (y/n)? y
-> 18.0
next (y/n)? y
-> 20.0
next (y/n)? n
nb termes supplementaires? 3
progression: 10.0 12.0 14.0 16.0 18.0 20.0 22.0 24.0 26.0
```

## 2 Progressions géométriques

Créer un répertoire de travail de nom `geometric` pour les classes de cette section.

Programmer de façon similaire une classe `ProgressionGeometrique` et sa classe `TestGeometrique`.

# 3 Surclasse abstraite et polymorphisme

Créer un répertoire de travail de nom `hierarchic` pour les classes de cette section.

Constater les fortes similitudes entre les classes `ProgressionArithmetique` et `ProgressionGeometrique`. En déduire une surclasse abstraite commune, soit `Progression`. La programmer ainsi que les nouvelles versions de `ProgressionArithmetique` et `ProgressionGeometrique`.

## Compilation modulaire

Le protocole des classes `ProgressionArithmetique` et `ProgressionGeometrique` n'a pas changé, de sorte que les programmes de test doivent toujours fonctionner, sans recompilation nécessaire. Vérifier cela en copiant dans le répertoire `hierarchic` le bytecode des classes de test `arithmetic/TestArithmetique.class` et `geometric/TestGeometrique.class` et en les exécutant directement.

## Polymorphisme

Constater également les fortes similitudes du code de test, les manipulations sont les mêmes, seul le type d'objet manipulé change. Programmer en conséquence une nouvelle classe principale `Test` qui demande à l'utilisateur le type de progression, ses caractéristiques, instancie la classe de Progression appropriée, et effectue les manipulations en la passant en paramètre (polymorphe) à une procédure `'void manip(Progression p)'` selon le schéma de code et d'exécution suivant :

```
public class Test {

 static void manip(Progression p) {
  // next (y/n)?, nb termes supplementaires?
 ...}

 public static void main (String[] args) {
  Progression p;
  // creation de la progression, type, raison, 1er terme?
  ...
  // manipulations de p
  manip(p);
 }
```

```
$ java Test
Progression arithmetique taper 1
Progression geometrique taper 2
? 2
premier terme? 10
raison? 2
next (y/n)? y
-> 20.0
next (y/n)? y
-> 40.0
next (y/n)? y
-> 80.0
next (y/n)? n
nb termes supplementaires? 3
progression: 10.0 20.0 40.0 80.0 160.0 320.0 640.0
```

# 4 Tester votre travail

Afin de savoir si votre code est de qualité et correspond à ce qui vous est demandé, il vous est possible de le tester. Pour cela, copier sur votre compte dans le répertoire `tp3` le répertoire `test` :

```
cp -r ~aetien/public/PPO/tp3/test .
```

Mettez vous dans votre répertoire `tp3/test` et exécutez la commande suivante pour tester la première version des progressions arithmétiques et géométriques :

```
./runTest.sh
```

Exécutez la commande suivante pour tester la deuxième version des progressions avec la hierarchie :

```
./runHierarchieTest.sh
```

Si votre code vérifie tous les critères de qualité évalués, vous aurez un message du genre `OK (4 tests)` pour la partie sans hiérarchie et `OK (7 tests)`, pour la partie avec. Sinon, vous aurez des messages vous indiquant les erreurs qui peuvent exister dans votre code.