L2 info S3
UE : Programmation Orientée Objets 1 - Java

TD3 - Classe abstraite ; interface

Objectifs

A la fin de ce TP, vous devez être capables de :

1. Intégrer des classes abstraites dans vos programmes Java.
2. Compléter vos classes pour qu'elles implémentent des interfaces.

1 Classe abstraite : suites récurrentes

On désire écrire un programme Java permettant de manipuler des suites récurrentes. Les suites pourront être soit des suites arithmétiques, soit des suites géométriques.

1.1 Rappels mathématiques

Les termes de la suite sont notés u_i. Le premier terme a pour indice 1.

Suite arithmétique

u_1 et r (valeur de la progression) sont donnés.
Les termes suivants sont calculés de façon récurrente : $u_{i+1} = u_i + r$

Calcul du $n^{ème}$ terme : $u_n = u_1 + (n-1)r$

Calcul de la somme des n premiers termes : $u_1 + u_2 + \ldots + u_n = \frac{n}{2}(u_1 + u_n)$

Suite géométrique

u_1 et q (valeur de la progression) sont donnés.
Les termes suivants sont calculés de façon récurrente : $u_{i+1} = qu_i$

Calcul du $n^{ème}$ terme : $u_n = u_1 q^{n-1}$

Calcul de la somme des n premiers termes : $u_1 + u_2 + \ldots + u_n = u_1 \frac{1-q^n}{1-q}$

1.2 Programmation Java

On vous demande d'écrire un programme Java qui comporte
- une classe abstraite `Suite`,
- les classes concrètes `SuiteArithm` (suites arithmétiques), `SuiteGeom` (suites géométriques)
- la classe `Test` comportant la méthode `main`.

Classe Suite

Cette classe comporte
- les attributs
  - `premier` : premier terme de la suite (u_1).
  - `pas` : valeur de la progression entre deux termes successifs de la suite (noté r ou q dans les rappels mathématiques).
- un constructeur `Suite(int,int)` qui initialise les 2 attributs `premier` et `pas`.
- les méthodes abstraites
  - `int valeurAuRangN(int)` qui permet de calculer u_n, n étant transmis en argument.
  - `int sommeAuRangN(int)` qui permet de calculer la somme des n premiers termes, n étant transmis en argument.

Classes SuiteArithm et SuiteGeom

Ces deux classes sont des classes filles de `Suite`. Elles sont concrètes.
- Donner un constructeur pour chacune des 2 classes.
- Donner le corps des méthodes `valeurAuRangN` et `sommeAuRangN`.
- Compléter la classe `SuiteArithm` avec la méthode `int calculRang(int)` où l'argument correspond à un certain u_n et le résultat renvoyé par la méthode est la valeur de l'indice n. Exemple : pour une suite arithmétique `s1` de premier terme 1 et de pas 3, `s1.calculRang(19)` renverra 7 car $19 = u_7$ pour cette suite. On supposera que l'argument transmis correspond bien à un terme u_n.

Indications :
- la classe `Math` a une méthode `static double pow(double a, double b)` qui renvoie a^b.
- faire attention aux arrondis :
  `int i=3/2*8;` affecte la valeur 8 à `i` alors que
  `int i=3*8/2;` affecte la valeur 12 à `i`.

Classe Test

Cette classe comporte la méthode `main` avec
- déclaration de deux suites `s1` et `s2`,
- instanciation de `s1` comme suite arithmétique et `s2` comme suite géométrique,
- appels pour `s1` et `s2` aux méthodes `valeurAuRangN` et `sommeAuRangN`.
- appel pour `s1` à la méthode `calculRang`.

Intérêt de la classe abstraite

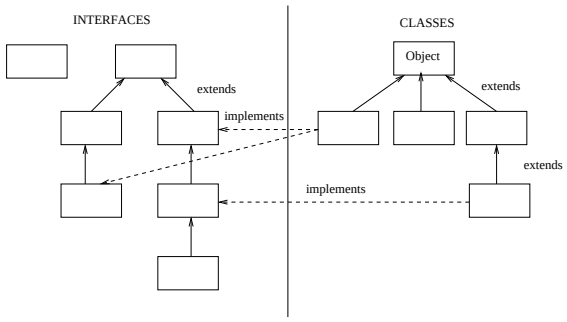
Dire pourquoi il n'était pas judicieux de programmer les 2 classes `SuiteArithm` et `SuiteGeom` da façon indépendante sans utiliser la classe abstraite `Suite`.

2 Interface : monstres et vampires

Les interfaces permettent de pallier à l'interdiction d'héritage multiple en Java.

Rappel :

> Les classes et les interfaces constituent deux hiérarchies distinctes qui peuvent être illustrées de la façon suivante :



Soit le programme java suivant :

```
interface Monster
{
  void menace();
}

interface DangerousMonster extends Monster
{
  void destroy();
}

interface Vampire
{
  void drinkBlood();
}

class DragonZilla implements DangerousMonster { ... }          /* question 1 */

class Bat implements Vampire { ... }                            /* question 1 */

class HorrorShow
{
  static void u(Monster b) { ... }                               /* question 2 */
  static void v(DagerousMonster d) { ... }                       /* question 2 */
  public static void main(String[] args)
  {
      DragonZilla if2 = new DragonZilla();
      ...                                                        /* question 3 */
      ...                                                        /* question 4 */
      Bat bt1 = new Bat();
      ...                                                        /* question 5 */
  }
}
```

Questions

On suppose que

— la méthode `menace()` permet d'afficher "brr..." à l'écran ;
— la méthode `destroy()` permet d'afficher "pan..." à l'écran ;
— la méthode `drinkBlood()` permet d'afficher "slurp..." à l'écran.

Dans les questions suivantes, donner le strict nécessaire (ne pas inventer d'attributs ou de méthodes supplémentaires).

1. Compléter les classes `DragonZilla` et `Bat` afin qu'elles respectent leurs déclarations.
2. Donner les instructions des méthodes `u` et `v` afin que toutes les méthodes applicables soient appelées (ordre quelconque).
3. Peut-on appliquer la méthode `u` à l'instance `if2` ? Justifier votre réponse.
   Si oui, écrire l'instruction correspondante.
4. Peut-on appliquer la méthode `v` à l'instance `if2` ? Si oui, écrire l'instruction correspondante.
5. Ecrire les instructions afin que toutes les méthodes de l'instance `bt1` soient appelées.